Antipattern Detection in Web Ontologies: an Experiment using SPARQL Queries

Catherine Roussey*, Oscar Corcho**, Ondřej Šváb-Zamazal***, François Scharffe****, Stephan Bernard*

*Irstea/Cemagref, 24 Av. des Landais, BP 50085, Aubiére, France
**Ontology Engineering Group, Universidad Politécnica de Madrid, Spain
***Knowledge Engineering Group, University of Economics Prague, Czech Republic
****LIRMM, Université de Montpellier, France

Abstract. Ontology antipatterns are structures that reflect ontology modelling problems because they lead to inconsistencies, bad reasoning performance or bad formalisation of domain knowledge. We propose four methods for the detection of antipatterns using SPARQL queries. We conduct some experiments to detect antipattern in a corpus of OWL ontologies.

1 Introduction

The concept of knowledge modelling pattern or ontology design pattern is used to refer to modelling solutions that allow solving recurrent knowledge modelling or ontology design problems, Presutti et al. (2008). Antipatterns are patterns that are ineffective or far from optimal in practice, representing worst practices about how to structure and design an ontology. There are several tools that can be used for the detection of antipatterns. Pellint [1] focuses on the detection and repair of antipatterns to improve ontology reasoning performance. Tools like Explanation Workbench described in Horridge et al. (2008), or SWOOP described in Kalyanpur et al. (2005), provide justifications of inconsistencies in ontologies based on the outputs of DL reasoners. However, all these contributions need a reasoner to provide their justifications.

Our antipattern detection methods implement a more general approach, that can work on any antipattern and can be applied without the use of a reasoner, something that is very useful with large ontologies and when the number of errors in an ontology is so large that the previous justification systems are not able to handle them properly, providing timeouts. To detect the selected antipattern, we have transformed it into sets of SPARQL queries. In general, antipatterns correspond to several queries, because they are abstract structures that can have several logical forms when expressed in Description Logics (DL). Moreover we have proposed several detection methods. We can switch on or off inferences before running SPARQL queries. We can also transform the original ontologies into a form where simpler SPARQL queries can be run.

This paper is structured as follows. Section 2 briefly describes the antipattern that will be used to run our experiments. Section 3 will describe the methods we have followed in order to run the experiments. Section 4 describes the experiment setup and the results of the

1. http://pellet.owldl.com/pellint

experimentation. Finally, Section 5 provides some conclusions to the work done, based on the experiment results, and outlines the next steps to be done in our work.

2 A sample antipattern OnlynessIsLoneliness (OIL)

A set of patterns commonly used by domain experts in their implementation of OWL ontologies are identified in Corcho et al. (2009). These patterns resulted in unsatisfiable classes or modelling errors, due to misuse or misunderstanding of DL expressions. In this section we will describe an antipattern which is the one that, our experience has shown, is easier to understand and debug by domain experts.

$$C_3 \sqsubseteq \forall R.C_1; C_3 \sqsubseteq \forall R.C_2; \qquad Disj(C_1, C_2); \tag{1}$$

$$C_3 \equiv \forall R.C_1; C_3 \sqsubseteq \forall R.C_2; \qquad Disj(C_1, C_2); \tag{2}$$

$$C_3 \equiv \forall R.C_1; C_3 \equiv \forall R.C_2; \qquad Disj(C_1, C_2); \tag{3}$$

The ontology developer created a universal restriction to say that C_3 instances can only be linked with property R to C_1 instances. Next, a new universal restriction is added saying that C_3 instances can only be linked with R to C_2 instances, with C_1 and C_2 disjoint. In general, this is because the ontology developer forgot the previous axiom in the same class or in any of the parent classes.

3 SPARQL-based detection of OIL antipattern

In this section we describe the different methods that we have elaborated in order to detect antipatterns in OWL ontologies by means of SPARQL queries, based on the usage of the PatOMat ontology pattern detection tool [2]. This tools is part of the PatOMat suite of tools, which is focused on the detection of patterns in ontologies and their transformation. This detection tool is based on Jena 2.6.2 [3] and Pellet 2.0.1 [4], and enables the processing of a set of SPARQL queries over a set of ontologies, producing a report in terms of numbers of patterns detected (SPARQL queries results) and details for each ontology. The axioms over which the pattern detection tool is run can be the axioms asserted in the ontologies, or a combination of asserted and inferred axioms.

We query OWL ontologies by means of a query language (SPARQL) that is agnostic about the underlying knowledge representation model of OWL: we are actually querying the RDF serialization of OWL. Other options that are available in the current state of the art for OWL ontology pattern matching and transformation are the OPPL language and its associated tools described by Iannone et al. (2009), or the more recent OWL querying syntax Terp [5], based on the OWL Manchester syntax. If SPARQL is the language dedicated to query RDF triples, OPPL and Terp are dedicated to query the RDF serialization of OWL expressions because they contain OWL constructs like subClassOf, ComplementOf, DisjointWith. Nevertheless to make

2. The release used for this paper is at: `http://eso.vse.cz/~svabo/patomat/detectionTool.zip`
3. `http://jena.sourceforge.net/`
4. `http://clarkparsia.com/pellet/`
5. `http://clarkparsia.com/weblog/2010/04/01/pellet21-terp/`

the construction of SPARQL queries easier, we develop a query translator that transforms an input query, using the SPARQL-DL abstract syntax defined in Sirin and Parsia (2007), into a SPARQL query.

Transforming antipatterns into SPARQL-DL queries is not a trivial task. For each antipattern, several SPARQL-DL queries are needed to detect antipattern occurrences in OWL class definition. The difficulties come from several points : - An antipattern can be associated to several logical formulae in DL syntax. For example, we presented 3 formulae for OIL antipatterns. - Some logical formulae are composed of several atomic axioms. [6] For example, the three formulae of the OIL antipattern contains three atomic axioms. - Ontology developer may have very different implementation style when designing an OWL ontology. For example, some developers may prefer to write long class definition. In that case, a class is defined by a conjunction of unamed classes: $C \sqsubseteq (\exists R.X) \sqcap (\forall R.Y) \sqcap ....$ Others may prefer to write short definitions. A class is defined by a set of atomic axioms: $C \sqsubseteq \exists R.X; C \sqsubseteq \forall R.Y; C \sqsubseteq ....$ Thus for an antipattern formula, an atomic axiom can be located at different places in the class definition. - An atomic axiom can belong to the class definition or can be inherited from a parent class definition. - An atomic axiom can be stated by the ontology developer or infered by a reasoner.

To build our queries, we first imagine different versions of each antipattern formulae using the SPARQL-DL abstract syntax. We try to imagine where an atomic axiom can be stated by the ontology developer in a class definition. We limit our imagination to class definitions that have at most four conjunctions. We embed in those queries some of the inferences that should be done by a reasoner. We take in account the fact that:

– disjoint axioms are symetric $Disj(C_1, C_2) \vDash Disj(C_2, C_1)$,

– disjoint axiom can be inferred from a logical negation $C_1 \sqsubseteq \neg C_2 \vDash Disj(C_1, C_2)$.

Then we automatically translate each SPARQL-DL queries into SPARQL ones. We also automatically generate SPARQL queries which merges all the different versions.

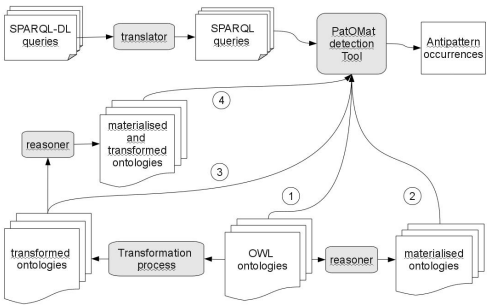

FIG. 1 – *The antipattern detection methods*

6. We defined an atomic axiom as a condition (necessary $\sqsubseteq$ or sufficient $\equiv$) associated to a named class C using at most one constructor ($\forall$, $\exists$, $\neg$ or $\sqcap$). All the parameters should be named. An example of atomic axiom can be $C \sqsubseteq \exists R.X$.

As shown in figure 1, we will describe now the four methods that we have followed in order to detect antipatterns in the ontology corpus:

-**Method 1**: Use of SPARQL Queries over Asserted OWL Ontology Axioms. In this approach, we take into account that SPARQL query engines per se do not consider inferences that can be done with OWL ontologies. However, our assumption is that there will be cases where ontologies cannot be processed by a reasoner or the reasoner results cannot be obtained in a reasonable time. This normally happens with large ontologies or with ontologies with a large number of errors.- **Method 2**: Use of SPARQL Queries over Materialised Inferences on OWL Ontologies. When it is possible to use a reasoner, we materialise all the inferences that can be done by an OWL reasoner on the ontologies and then run SPARQL queries over the resulting ontologies, called materialised ontologies. - **Method 3 and 4**: Use of SPARQL Queries over Transformed OWL Ontologies. Due to the complexity of creating a large number of SPARQL queries for an antipattern and to the fact that different ontology developers may have different *implementation styles*, we propose to follow a two step process where we apply transformations before executing the queries. Transformations have two goals: to harmonise the implementation style of the ontology and to simulate some of the axioms infered by reasoner. The current transformations that we apply are: - When the ontology contains $C_1 \equiv C_2$ where C_1 and C_2 are named classes, we add two new axioms $C_1 \sqsubseteq C_2$ and $C_2 \sqsubseteq C_1$. - When a named class is defined by conjunction of named or unamed classes, we split it into several simpler axioms. Take as example the class definition: $C \sqsubseteq X \sqcap Y$, in that case we add two axioms $C \sqsubseteq X$ and $C \sqsubseteq Y$. - When a parent class contains an axiom, we add it also in its direct child class. Take as example the definition of the class: $C_1 \sqsubseteq \exists R.X$. If $C_{1.1}$ is a direct child of C_1, $C_{1.1} \sqsubseteq C_1$, we add the axiom $C_{1.1} \sqsubseteq \exists R.X$. At that point, this transformation is not repeated over the class hierarchy.

In this case, we have explored the behaviour of the SPARQL query checking method both on the asserted ontology after transformation and on the materialised ontology (also after transformation).

4 Finding antipatterns in real-world ontologies

In this section, we describe the results of our experiments with a corpus of ontologies extracted from those publicly available on the Web and indexed by the Watson semantic search engine [7]. We will first describe the ontology corpus, and then the results of applying the different methods described in Section 3 to this ontology corpus.

4.1 Experiments

The ontologies used in our experimentation comes from our experience in ontology debugging task. Five of them have already been used for the creation and update of the antipattern catalogue presented in Corcho et al. (2009). It contains the HydrOntology (which has 159 classes whose 114 are unsatisfiables), the Forestal Ontology (which has 93 classes whose 62 are unsatisfiable), the Tambis ontology (which has 395 classes whose 112 are unsatisfiable), the Sweet Numeric ontology (which has 2364 classes whose 2 are unsatisfiable) and the University ontology of the MIND Lab (which has 29 classes whose 7 are unsatisfiable). Notice

7. http://watson.kmi.open.ac.uk/

that in our experiment Hydrontology and the Tambis ontologies cannot be processed by the Pellet reasoner in a reasonable time.

We made the following experiments over the set of ontologies, using the antipattern detection methods described in Section 3:

1. SP: Search in the original ontologies (only with asserted axioms) using SPARQL queries and no inference.
2. SP+R: Search in the materialised ontologies (asserted and inferred axioms) using SPARQL queries after applying a reasoner (Pellet).
3. SP_Trans: Apply transformations on the original ontologies and search (only with asserted axioms) using SPARQL queries and no inference.
4. SP_Trans+R: Apply transformations on the original ontologies and search in the materialisation of these harmonised ontologies.

In these experiments we also use the keyword MANUAL to refer to the manual detection process using the basic debugging tools provided by ontology editors. This detection method sets a baseline with respect to what can be detected on the current state of the art.

We have also run experiments, based on the previous ones, to evaluate the precision of the antipattern detection process. We have analysed manually each of the ontologies in our set and have assigned to each antipattern occurrence one of the following three values:

– TI (True Inconsistency): the antipattern occurrence participates in the unsatisfiability of classes or the modelling error.
– UI (Unknown Inconsistency): the antipattern occurrence may be linked to the unsatisfiability of classes or modelling error, but the evaluator is not sure about it.
– FI (False Inconsistency): the antipattern occurrence does not participate in the unsatisfiability of classes or modelling error.

4.2 Results: OIL detection

The OIL pattern is composed of 3 atomic axioms. We have presented 3 formulae but more formulae are possible, depending of the implementation style of the ontology developer. See the ontology antipattern web site [8] for more OIL antipattern formulae. For these formulae, we imagine that a class definition can be composed of two conjunctions parts. We defined 84 SPARQL queries. The results presented in table 1 for the detection of the antipattern are

method	nb of results	nb of TI	nb of UI	nb of FI	nb of onto
manual	8				3
SP	2	2	0	0	2
SP+R	2	2	1	0	2
SP_Trans	2	2	0	0	2
SP_Trans+R	72	6	66	0	2

TAB. 1 – *OIL antipattern detection.*

unexpected. We notice that the disjoint atomic axiom was not detected because it is infered by

8. https://sites.google.com/site/ontologyantipattern/

the reasoner. And using a reasoner produce unexpected antipattern occurrences. Thus at that time any of our detection methods is good enought to detect OIL antipattern. We should limit our detection method to the beginning of the OIL pattern without the disjoint axiom.

5 Conclusion and future work

In this paper we have shown how OIL antipattern can be detected using different methods that are based on the use of SPARQL queries, OWL reasoners and transformation tools. In many cases, these antipattern detection methods are very sensitive to the *implementation style* of the ontology developer. Moreover reasoners cannot be used due to bad response time and unexpected results. Our future work will focus on the refinement of the methods that we have proposed in this paper to improve the detection results. We will also try to detect new antipatterns.

References

Corcho, O., C. Roussey, L. M. Vilches Blázquez, and I. Pérez (2009). Pattern-based OWL ontology debugging guidelines. In *Workshop on Ontology Patterns (WOP 2009), collocated with the 8th International Semantic Web Conference (ISWC-2009).*, CEUR Workshop proceedings, pp. 68–82.

Horridge, M., B. Parsia, and U. Sattler (2008). Laconic and precise justifications in OWL. In *Proceedings of ISWC*, pp. 323–338.

Iannone, L., A. L. Rector, and R. Stevens (2009). Embedding knowledge patterns into OWL. In *Proceedings of ESWC*, pp. 218–232.

Kalyanpur, A., B. Parsia, E. Sirin, and J. Hendler (2005). Debugging unsatisfiable classes in OWL ontologies. *Journal of Web Semantics 3*(4), 268–293.

Presutti, V., A. Gangemi, S. David, G. de Cea, M. Suárez-Figueroa, E. Montiel-Ponsoda, and M. Poveda (2008). NeOn deliverable d2. 5.1. a library of ontology design patterns: reusable solutions for collaborative design of networked ontologies. *NeOn Project. http://www. neon-project. org*.

Sirin, E. and B. Parsia (2007). SPARQL-DL: SPARQL query for OWL-DL. In *3rd OWL: Experiences and Directions Workshop (OWLED2007)*.

Résumé

Les antipatrons de conception d'ontologies sont des structures abstraites qui reflètent des problèmes de modèlisation. Ils peuvent mener à des incohérences logiques, de mauvaises performances des moteurs d'inférences, ou des formalisations inadéquates des connaissances du domaine. Il est donc important de détecter les antipatrons, pour corriger les ontologies. Dans cet article nous proposons quatre méthodes de détection d'antipatrons à partir de requêtes SPARQL. Pour évaluer nos méthodes, nous avons testé nos requêtes SPARQL sur un ensemble d'ontologies réelles.