HAL
open science

Pattern-based ontology transformation service

Ondrej Sváb-Zamazal, Vojtech Svátek, François Scharffe

To cite this version:

Ondrej Sváb-Zamazal, Vojtech Svátek, François Scharffe. Pattern-based ontology transformation service. Proc. 1st IK3C international conference on knowledge engineering and ontology development (KEOD), Oct 2009, Funchal, Portugal. pp.210-223. hal-00822899

HAL Id: hal-00822899

https://inria.hal.science/hal-00822899v1

Submitted on 15 May 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

PATTERN-BASED ONTOLOGY TRANSFORMATION SERVICE

Ondřej Šváb-Zamazal[1,2], Vojtěch Svátek[1]
[1]*University of Economics, Prague, Czech Republic*
ondrej.zamazal@vse.cz, svatek@vse.cz

François Scharffe[2]
[2]*INRIA & LIG, Montbonnot, France*
francois.scharffe@inrialpes.fr

Keywords: Ontology transformation, Ontology alignment, Pattern detection, Correspondence patterns.

Abstract: Many use cases for semantic technologies (eg. reasoning, modularisation, matching) could benefit from an ontology transformation service. This service is supported with ontology transformation patterns consisting of corresponding ontology patterns capturing alternative modelling choices, and an alignment between them. In this paper we present the transformation process together with its two constituents: a pattern detection and an ontology transformation process. The pattern detection process is based on SPARQL and the transformation process is based on an ontology alignment representation with specific extensions regarding detailed information about the transformation.

1 INTRODUCTION

On-demand ontology transformation *inside a formalism* such as OWL can be useful for semantic applications. The same conceptualization can be modeled in diverse ways; (parts of) an ontology thus can be transformed from one *modeling choice* to another, taking advantage of ontology patterns for eg. n-ary relations, specified values or naming variations. Although the strictly formal semantics may change, the *intended meaning* should be preserved.

In (Svab-Zamazal et al., 2009) are described three use cases:

- **Reasoning.** Some features of ontologies cause performance problems for certain reasoners. Having information about these features, possibly gathered via machine learning methods, we can transform parts of ontologies with such problematic entities.
- **Modularization.** Modular ontologies are a prerequisite for effective knowledge sharing. However, if the source and target ontology are modeled using different styles (such as property- vs. relation-centric), the user faces difficulties when choosing fragments to be imported. modeling style of the various modules.
- **Matching.** Most *ontology matching* (OM) tools deliver simple entity-to-entity correspondences. When applying lexical methods on names of entities, they ignore matching of complex structures. Complex matching can be mediated by correspondence patterns, which however most OM tools do not support. Attempting to transform, prior to matching, an ontology to its variant using transformation patterns, could thus help the OM tools.

These use cases share an *ontology transformation service* that makes use of *ontology transformation patterns*. In this paper we present details about the ontology transformation service, based on the detection and transformation of patterns in ontologies. We clearly distinguish the modeling of ontology patterns, used for the detection phase of the transformation service, and the transformation itself. The link between the ontology patterns and the transformation is realized through the use of an alignment between the pattern to be detected and the pattern resulting from the transformation.

The rest of the paper is organized as follows: Section 2 presents the overall work-flow of ontology transformation service. Section 3 details the ontology pattern detection phase using an illustrative example. Section 4 details the transformation phase using ontology transformation patterns, following the same example. The paper is wrapped up with Related work, Conclusions and Future Work.

2 ONTOLOGY TRANSFORMATION SERVICE WORK-FLOW

This section presents the work-flow (See Figure 1) of the ontology transformation service. The transformation service takes as input an ontology O_1 and an ontology transformation pattern. It outputs a new ontology O_2 resulting from applying an ontology transformation pattern on given ontology O_1. An ontology transformation pattern consists of an ontology pattern A, its counterpart ontology pattern B, and an alignment between them. The ontology pattern A and the ontology pattern B represent the same conceptualization, but modeled in two different ways. The alignment between the two ontology patterns is represented using an ontology alignment language (Euzenat et al., 2007), with several extensions detailed in Section 4. In a first step, an ontology pattern A is detected in the ontology O_1. This procedure is further detailed in Section 3. If this ontology pattern matches, the ontology O_1 is transformed into an ontology O_2 given the ontology transformation pattern and generated transformation operations. Section 4 presents this second step in more details.

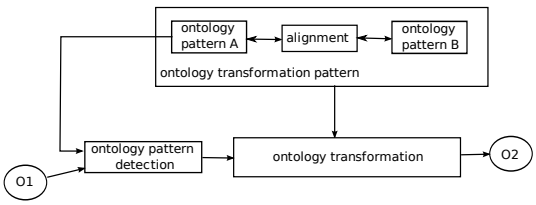

Figure 1: Ontology transformation service workflow.

During the ontology transformation process, many ontology transformation patterns from the input library may be detected and applied. Moreover, an ontology transformation pattern may be applied many times if its ontology pattern A was detected a number of time in the ontology.

Generally, we do not expect full automation of the ontology transformation, as there often would be numerous alternatives that only a human could evaluate. We thus aim at *semi-automatic* transformation (with appropriate user interface) only. In fact, the user will be shielded from this transformation service with the system where (s)he could opt diverse level of automation of the whole process eg. (not)selecting and/or (not)changing the ontology patterns applied, (not)modifying the ontology transformation patterns if the transformation results are not satisfactory etc. In this paper we concentrate merely on the core transformation service while issues such as ontology patterns ordering, checking the consistency of the newly created ontology (a validation step) and user interaction are future work.

3 PATTERN DETECTION

In this section we detail the ontology pattern detection phase. This phase takes as input the ontology pattern A of an ontology transformation pattern and tries to match this ontology pattern in the ontology O_1. Each ontology pattern A consists of three intertwined aspects: aspects related to the ontology structure, aspects related to the logical axioms such as disjointness, and aspects related to entities names. For detection we can use the SPARQL language[1].

We illustrate an ontology pattern detection on a simple example of ontology pattern A where a class is defined using an existential restriction[2]. Let us assume the following conceptualization of "PresentedPaper":

O_1: $Document \sqsupseteq Paper \sqsupseteq PresentedPaper \equiv Paper \sqcap \exists hasStatus.\{Accept\}$

"PresentedPaper" is restricted on the value "Accept" for the property "hasStatus". This is a common conceptualization (eg. man with married state, paper with status poster). An alternative to this ontology pattern is presented in Section 4. The SPARQL query for detection of this ontology pattern is the following:

```
SELECT ?y ?p ?a
WHERE {
   ?y rdfs:subClassOf _:b.
  _:b owl:onProperty ?p.
  _:b owl:hasValue ?a.
FILTER ( !isBlank(?y) )}
```

There are several drawbacks using the SPARQL language for such a detection, see Section 5. In our current implementation, we use the SPARQL query engine from the Jena framework [3].

The following example of ontology pattern deals with "specified values" representing specified collection of values expressing 'qualities', 'attributes', or 'features', see the left-hand side in Figure 3. The corresponding SPARQL query is:

```
SELECT distinct ?y ?c1 ?c2
WHERE {
  ?c1 rdfs:subClassOf ?y.
  ?c2 rdfs:subClassOf ?y.
  ?c1 owl:disjointWith ?c2.
```

[1] http://www.w3.org/TR/rdf-sparql-query/

[2] All examples come from the conference organization domain.

[3] http://jena.sourceforge.net/

```
FILTER (
  !isBlank(?c1) &&
  !isBlank(?c2) &&
  !isBlank(?y))}
```

In order to better detect the "specified values" ontology pattern, a lexical constraint on the entities name is used. An average token-based similarity measure c is computed as the ratio of the number of mutually disjoint entities $(?c1, .., ?cn)$ that are subclasses of $?y$ sharing a token with $?y$ over the number of all distinct such entities $(?c1, ..., ?cn)$. If c is higher than certain threshold it is taken as an instance of the "specified values" ontology pattern. As we can see in Figure 3 the token "topic" is present in the superclass as well as in both subclasses "MultimediaTopic" (MT) and "ComputerNetworkTopic" (CNT).

The next example of ontology pattern A deals with "n-ary relations". In OWL there is direct support for representing a binary relation. However, it is sometimes more natural to link more than two individuals in one relation. In order to detect this kind of ontology pattern, we define a structural bunch (Svab-Zamazal and Svatek, 2009) which can be found with the following SPARQL query:

```
SELECT ?x ?p ?y ?q ?r ?z ?w
WHERE {
  ?p rdfs:domain ?x.
  ?p rdfs:range ?y.
  ?q rdfs:domain ?y.
  ?q rdfs:range ?z.
  ?r rdfs:domain ?y.
  ?r rdfs:range ?w
FILTER (?q!=?r)}
```

Additionally, a lexical constraint similar to the aforementioned one can be used to refine the detection: an average token-based similarity measure c is computed as the ratio of the number of entities from a structural bunch that share a token with $?p$ over the number of all distinct entities from the structural bunch except $?y$ and $?p$. If c is higher than a certain threshold it is taken as an instance of "n-ary relation" ontology pattern.

The three ontology patterns presented in this section can be classified as logical patterns. We have included them into the ontology transformation pattern library (ie. for each of them we have an ontology pattern A, an ontology pattern B and an alignment). In future work we will extend the ontology transformation library with other ontology patterns like eg. name patterns (Svab-Zamazal and Svatek, 2008) capturing awkward naming style.

Once an ontology pattern is detected, the corresponding transformation can be applied as detailed in Section 4.

4 ONTOLOGY TRANSFORMATION

In this section we detail the transformation phase of the ontology transformation service. This phase takes as input the result of an ontology pattern detection, ie. bound variable(s) from the corresponding SPARQL query, and the ontology transformation pattern related to the detected ontology pattern A. Initially, the entire ontology O_1 is copied into a new ontology O_2. Afterwards, a set of specific transformation operations are generated according to an appropriate ontology transformation pattern. The new ontology O_2 is then transformed applying the operations. We will detail how to generate those transformation operations below in this section. We will also present at the end of this section three examples illustrating various aspects of the transformation process.

Note that a given ontology pattern A can be transformed in more than one way, ie. more than one ontology pattern B and appropriate alignment for each ontology pattern A are possible. This issue is our future work. The result of the transformation step is a new ontology O_2. It can be useful to enable reverse transformation from the ontology O_2 back to the ontology O_1. This can be realized using OWL-2 annotations materializing information that can be lost during a transformation.

There are three general rules how to generate specific transformation operations:

1. if there is a correspondence between entities of ontology patterns A and B (entity-to-entity) in the alignment then REMOVE the old entity and ADD the new entity with proper name and proper type[4]. There is an instruction how to name new entity in the alignment.
2. If there is no existing correspondence from an entity in ontology pattern A to any entity of ontology pattern B in the alignment then REMOVE this entity.
3. If there is no existing correspondence from an entity in ontology pattern B to any entity of ontology pattern A in the alignment then ADD this entity.

As a result, several ADD/REMOVE operations are generated to be applied on copied version of O_1. The REMOVE operation must be done carefully in order to avoid removing an entity which is used in another part of the original O_1. Consistency checking and user validation need to be performed each time as

[4]Entities types can be different in case there is a correspondence between heterogeneous entities, eg. between a class and a relation.

an ontology pattern is applied in order to avoid this kind of problem.

Alignments between ontology patterns A and B of the ontology transformation patterns are expressed using an extension of the ontology alignment format (Euzenat et al., 2007). This format allows to model complex correspondence arising when trying to relate existing ontology patterns. This format also has the advantage to be supported by many ontology matching systems[5].

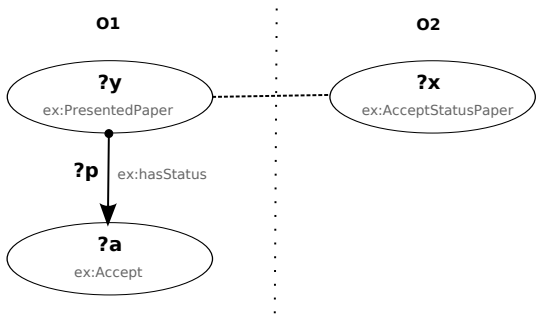

Figure 2: Ontology pattern: class using an existential restriction.

Next, we illustrate a transformation on the running example. The ontology transformation pattern is depicted in Figure 2. The corresponding alignment is represented below in the extended alignment format[6]:

```
<alignment:Cell rdf:about="class-by-attribute-value-cell">
  <alignment:relation rdf:resource="&alignment;equivalence"/>
  <alignment:entity1>
    <alignment:Variable>
      <rdf:type resource="&alignment;Class"/>
      <alignment:var_id>?x</alignment:var_id>
      <transf:resourceName>
        <rdf:Seq>
          <rdf:li>?a</rdf:li>
          <rdf:li>patomat:head_noun(?p)</rdf:li>
          <rdf:li>patomat:head_noun(?y)</rdf:li>
        </rdf:Seq>
      </transf:resourceName>
    </alignment:Variable>
  </alignment:entity1>
  <alignment:entity2>
    <alignment:Variable>
      <rdf:type resource="&alignment;Class"/>
      <alignment:var_id>?y</alignment:var_id>
      <alignment:attributeValueCondition>
        <alignment:Restriction>
          <alignment:onProperty>
           <alignment:Variable>
            <rdf:type resource="&alignment;Relation"/>
            <alignment:var_id>?p</alignment:var_id>
           </alignment:Variable>
          </alignment:onProperty>
          <alignment:value>
           <alignment:Variable>
            <rdf:type resource="&alignment;Instance"/>
            <alignment:var_id>?a</alignment:var_id>
           </alignment:Variable>
          </alignment:value>
        </alignment:Restriction>
      </alignment:attributeValueCondition>
    </alignment:Variable>
  </alignment:entity2>
</alignment:Cell>
```

This example is a common pattern in ontology alignment (Scharffe, 2009) extended with transformation-specific information. In order to bind variables between ontology patterns and the alignment we introduce the class `alignment:Variable`. Variables in a correspondence of the alignment format are bound to variables of an ontology pattern A or B using `alignment:var_id`. The alignment format extension can be used to modify attribute values but not the resources names themselves. We thus make use of another extension `transf:resourceName` and external functions enabling this transformation.

The transformation function `transf:resourceName` contains information how to name a resource in ontology pattern B. This function concatenates the list of entity names given in $< rdf : Seq >$. In the example above, we use the specific lexical function `patomat:headnoun(?a)` which returns the head noun of an entity name on which all other tokens are dependent. The head noun detection was introduced in (Svab-Zamazal and Svatek, 2008). Regarding property names, we can use several heuristics how to find a main term, eg. for "hasStatus" it would be "status" where "has" is prefix.

In this case, the generation of the transformation operations gives the following result: REMOVE the class "PresentedPaper" and ADD the class with name "AcceptStatusPaper". These operations result from applying rule #1. "Accept" is the name of instance $?a$, "Status" is the head_noun of property $?p$, and "Paper" is the head_noun of class $?y$. Then, "hasStatus" and "Accept" will be removed with REMOVE operations. There are other alternatives regarding naming of the new entity "PresentedPaper". These alternatives can be captured with the next `transf:resourceName` function:

```
<transf:resourceName>
  <rdf:Seq>
    <rdf:li>patomat:head_noun(?y)</rdf:li>
    <rdf:li>"With"</rdf:li>
    <rdf:li>patomat:head_noun(?p)</rdf:li>
    <rdf:li>?a</rdf:li>
  </rdf:Seq>
</transf:resourceName>
```

[5]See the ontology alignment initiative: http://oaei.ontologymatching.org

[6]http://alignapi.gforge.inria.fr/language.html

This will cause naming the entity as "PaperWithStatusAccept". These options can be offered to the final user for selecting the most appropriate one.

The "specified values" ontology transformation pattern is depicted in Figure 3. According to the transformation rules, the following transformation operations will be generated: REMOVE and ADD operations for each subclass. Because there are correspondences between heterogeneous entities, classes will be removed and instances will be added.

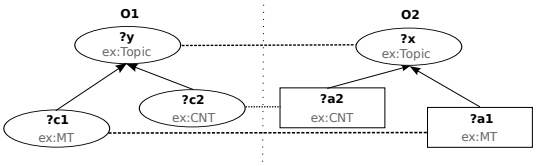

Figure 3: Pattern: specified values.

The "Nary relations" ontology transformation pattern is depicted in Figure 4. According to the transformation rules, the following transformation operations will be generated: REMOVE entity ?p, ?q, and ?v because of rule #2 and REMOVE entity ?r, ?w, and ADD new entity name according to original names of ?r and ?w in the following way:

```
<transf:resourceName>
  <rdf:Seq>
    <rdf:li>patomat:head_noun(?r)</rdf:li>
    <rdf:li>patomat:head_noun(?w)</rdf:li>
  </rdf:Seq>
</transf:resourceName>
```

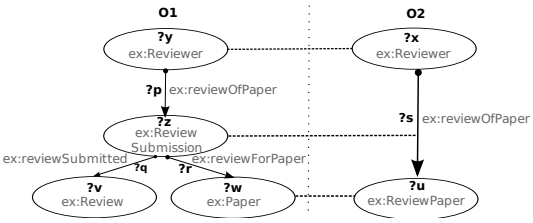

Figure 4: Ontology pattern: specified values.

5 RELATED WORK

Regarding ontology pattern detection, there are two related aspects: ontology patterns representation, and patterns detection. Regarding patterns in ontologies, here presented ontology patterns are based on results of *Semantic Web Best Practices and Deployment Working Group*[7] (SWBPD). There are further activities in this respect like *ontology design patterns* (ODP)[8]. The SWBPD concentrates on *logical patterns* which are domain-independent, the ODP considers many diverse kinds of ontology design patterns (incl. logical patterns, content patterns, reasoning patterns etc.). So far we did not directly reuse ODP patterns but we plan to do so in the close future.

While the purpose of those two activities is to provide ontology designers with the best practices on how to model certain situations, we are interested in detecting these ontology patterns. On the one hand ontology patterns can emerge by chance or they can be used intentionally by the ontology designer. In the latter case, detection of ontology patterns should be easier. In both cases, since we have in mind an ontology transformation we always take an ontology pattern and its one or more alternative variants.

SPARQL enables us to match structural aspects of ontology patterns by specifying a graph pattern with variables. But SPARQL is a query language for RDF. However ontology patterns are rather DL-like conceptualizations. Therefore we have to consider a translation step between DL-like conceptualizations and the RDF representations which is not unique. In order to overcome this kind of issue we could use some an OWL-DL aware query language, eg. SPARQL-DL(Sirin and Parsia, 2007). However this language does not support some specific DL constructs e.g. *restriction* and it is not fully implemented yet. Next, we should consider not only asserted axioms but also hidden ones. This could be realized using a reasoner which could materialize all hidden axioms. Furthermore the SPARQL language is not sufficient for specific lexical constraints (such as synonymy or hyperonymy). We need to either make some additional checking (post-processing), or alternatively to implement a specific SPARQL FILTER function. Such FILTER functions could make the SPARQL language quite expressive however they are also usually computationally expensive (Pérez et al., 2006). It raises a question of right balance between the expressivity of the query language (here it holds for working with synonyms/hyperonyms in SPARQL query) and computational efficiency. By now, we use a two-phase process for detection of ontology patterns: a SPARQL query for the structural aspects and then a post-processing of the results for lexical constraints.

In (Svab-Zamazal and Svatek, 2008), the authors worked on the detection of ontology name patterns. These name patterns will be included into the ontology transformation pattern library as a specific kind of negative patterns. Furthermore, we reuse several concepts for detection and transformation (lexical processing).

[7] http://www.w3.org/2001/sw/BestPractices/

[8] See e.g. http://ontologydesignpatterns.org

Regarding the transformation part of our work, an immediate solution would be to use XSLT. However, XSLT transformations are not directly applicable to RDF because of its alternative representations. XSPARQL (Akhtar et al., 2008) overcomes this limitation by combining SPARQL with XSLT. XSPARQL constitutes an alternative to detecting and transforming ontology parts as we propose in this paper. It however mixes the detection and transformation parts. As already mentioned in the introduction of this paper we need to keep a clear distinction between the pattern detection and the transformation process. For this reason, we also do not directly base on other related approach, the OPPL(Iannone et al., 2008), which is a language for detecting and transforming ontology patterns. Furthermore there is no support for lexical aspect in this last approach.

Another related approach is GeRoMeSuite (Kensche et al., 2007), a system for models management (such as XML Schema, OWL, SQL) that implements matching, merging and composing of models. It also supports many kind of transformations such as transformation of data, identifiers, dates, and transformation between different formalisms. This work however does not consider transformations between heterogeneous conceptualizations and detection of ontology patterns. Furthermore, a lot of attention has been paid to transformation between different modeling languages, transformation based on meta-modeling using UML, and specifically transformation of datamodels (Omelayenko and Klein, 2003).

6 CONCLUSIONS AND FUTURE WORK

In this paper we have presented the details of an ontology transformation service with its two phases: ontology pattern detection and ontology transformation using ontology transformation patterns. The proposed service makes use of the existing formalisms SPARQL and the alignment format, combined with transformation rules in order to detect and transform parts of ontologies. This approach allows to reuse ontology patterns and correspondence/transformation patterns developed by third parties.

An implementation of the core service is our imminent future work. Further we will consider ontology transformation from a system viewpoint, incl. user interaction, incremental selection of patterns, consistency checking of the newly created ontology etc. Furthermore we will extend the ontology transformation pattern library with other ontology patterns, e.g. name patterns and other patterns based on the ODP portal. We also plan to further experiment with ontology pattern detection using diverse approaches presented in the previous section.

ACKNOWLEDGEMENTS

The research was partially supported by the IGA VSE grant no.20/08 'Evaluation and matching ontologies via patterns'. The authors would also like to thank Jérôme Euzenat for consultations during writing this text.

REFERENCES

Akhtar, W., Kopecky, J., Krennwallner, T., and Polleres, A. (2008). XSPARQL: Traveling between the XML and RDF worlds and avoiding the XSLT Pilgrimage. In *ESWC-08*.

Euzenat, J., Scharffe, F., and Zimmermann, A. (2007). D2.2.10: Expressive alignment language and implementation. Project deliverable 2.2.10, Knowledge Web NoE (FP6-507482).

Iannone, L., Egana, M., Rector, A., and Stevens, R. (2008). Augmenting the Expressivity of the Ontology Pre-Processor Language. In *OWLED 2008*.

Kensche, D., Quix, C., Chatti, M. A., and Jarke, M. (2007). GeRoMe: A Generic Role Based Metamodel for Model Management. In *Journal on Data Semantics*.

Omelayenko, B. and Klein, M., editors (2003). *Knowledge Transformation for the Semantic Web*. IOS press, Amsterdam (NL).

Pérez, J., Arenas, M., and Gutierrez, C. (2006). Semantics and Complexity of SPARQL. In *International Semantic Web Conference*, pages 30–43.

Scharffe, F. (2009). *Correspondence Patterns Representation*. PhD thesis, University of Innsbruck.

Sirin, E. and Parsia, B. (2007). SPARQL-DL: SPARQL Query for OWL-DL. In *OWLED2007*.

Svab-Zamazal, O. and Svatek, V. (2008). Analysing Ontological Structures through Name Pattern Tracking. In *EKAW 2008*.

Svab-Zamazal, O. and Svatek, V. (2009). Towards Ontology Matching via Pattern-Based Detection of Semantic Structures in OWL Ontologies. In *Znalosti 2009*.

Svab-Zamazal, O., Svatek, V., David, J., and Scharffe, F. (2009). Towards Metamorphic Semantic Models. In *In Poster session at ESWC-09*.