Detection and Transformation of Ontology Patterns

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

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

Abstract. As more and more ontology designers follow the pattern-based approach, automatic analysis of those structures and their exploitation in semantic tools is becoming more doable and important. We present an approach to ontology transformation based on transformation patterns, which could assist in many semantic tasks (such as reasoning, modularisation or matching). Ontology transformation can be applied on parts of ontologies called ontology patterns. Detection of ontology patterns can be specific for a given use case, or generic. We first present generic detection patterns along with some experimental results, and then detection patterns specific for ontology matching. Furthermore, we detail the ontology transformation phase along with an example of transformation pattern based on an alignment pattern.

1 Introduction

On-demand ontology transformation *inside a formalism* such as OWL[1] can be useful for many semantic applications. The motivation for transformation is that the same conceptualization can be formally modeled in diverse ways; (parts of) an ontology thus can be transformed from one *modeling choice* to another, taking advantage of logical ontology patterns, such as those published by W3C and referring to e.g. 'n-ary relations' [5] or 'specified values'. Although the strictly formal semantics may change, the *intended meaning* of the conceptualisation should be preserved.

In [15] 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 pre-requisite for effective knowledge sharing, often through *importing*. 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.

[1] `http://www.w3.org/TR/owl2-primer/`

– **Matching.** Most *ontology matching* (OM) tools deliver simple entity-to-entity correspondences. Complex matching can be mediated by alignment (originally called 'correspondence') patterns [9], 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 our pattern detection and transformation approach. We propose generic detection based on SPARQL query and sketch the specific detection within ontology matching context. Ontology transformation is based on transformation patterns which are close to alignment patterns.

The rest of the paper is organized as follows. Section 2 presents the overall workflow of ontology transformation, followed with an illustrative example. Section 3 details the generic ontology pattern detection along with results of experiment. Next, we describe specific detection method in context of ontology matching. Section 4 presents the relationship between transformation pattern and alignment pattern and gives an example. The paper is wrapped up with Related work, Conclusions and Future Work.

The paper is an extended version of [17]; the major novelty is in the pattern detection method that takes account of the target ontology and applies clustering to the pre-matched correspondences.

2 Ontology Transformation Process

2.1 Workflow of Ontology Transformation

This section presents the workflow of the ontology transformation. The transformation as such takes as input an ontology $O1$ and an ontology transformation pattern. It outputs a new ontology $O1'$ resulting from applying an ontology transformation pattern on $O1$. An ontology transformation pattern consists of an ontology pattern A, its counterpart ontology pattern B, and a pattern transformation between them.

Definition 1 (Ontology Pattern). Ontology Pattern *consists of:*

- *Mandatory non-empty set E of* entity declarations*, i.e. axioms with* rdf:type *property, in which entity* placeholders *are used instead of concrete entities.*[2]
- *Optional set Ax of* axioms *asserting facts about entities from E.*
- *An optional* naming pattern *NP capturing the naming aspect of the ontology pattern relevant for its detection.*

Definition 2 (Pattern Transformation). Pattern Transformation *(PT) consists of:*

[2] Placeholders are used in transformation patterns in general.

– *Mandatory set LI of* links*, where a link can be* either *an equivalence correspondence*[3] *or an extralogical link* eqAnn *between an annotation literal and a real entity*[4] *or a link* eqHet *between heterogeneous entities.*
– *Optional set ENP of* entity naming transformation patterns.

The ontology pattern A and the ontology pattern B typically represent the same conceptualization modeled in two different ways. The transformation pattern captures information on which entities should be transformed and how. This is similar to alignment patterns to some extent, see Section 4.

In Figure 1 you can see this three-step workflow of ontology transformation (for more details see [16]). Rectangle-shaped boxes represent RESTful services, while ellipse-shaped boxes represent input/output data.[5] Ontology transformation is broken up into three basic services:[6]

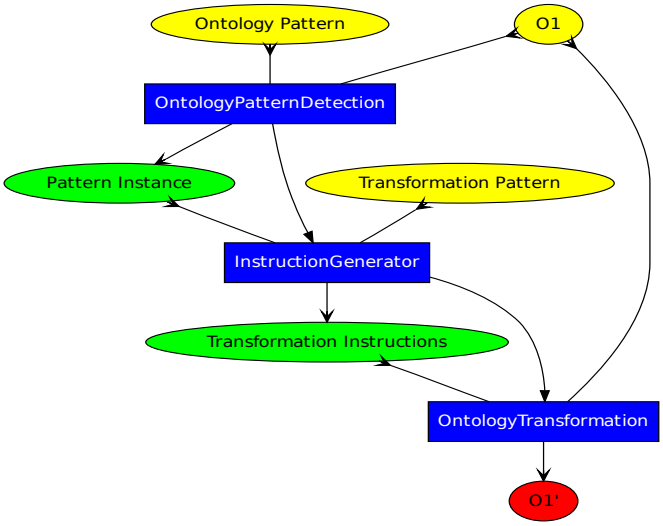

Fig. 1. Ontology Transformation Workflow

[3] Currently, we do not consider further correspondence relations such as disjointWith, subClassOf etc. for specifying specific relation between old and new version of entity (i.e. versioning). It can be considered in future work.
[4] An annotation literal is an entity that is only used for annotation purposes, while a real entity is real in this sense.
[5] In colours, blue boxes represent RESTful services; yellow ones represent input; green ones represent output which are in next step input; red ones represent output.
[6] Accessible via an HTML user interface at `http://owl.vse.cz:8080/`.

OntologyPatternDetection[7] outputs binding of placeholders in XML. It takes transformation pattern with its ontology patterns and particular ontology on input. This service internally automatically generates query based on ontology pattern and executes it. Structural/logical aspect is captured as SPARQL query and naming constraint is specifically dealt with based on description within ontology pattern. By now, this is partly implemented. This will be fully implemented with support of a Manchester syntax in SPARQL which will be available in new release of Pellet at the end of March, 2010. Details about two kinds of detection are in section 3.

InstructionGenerator[8] outputs particular transformation instructions in XML. It takes particular binding of placeholders and transformation pattern[9] on input. Transformation instructions are generated according to transformation pattern and pattern instance.

OntologyTransformation[10] outputs transformed ontology. It takes particular transformation instructions and particular ontology on input. This service is partly based on Ontology Pre-Processor Lanaguage (OPPL) [3] useful for manipulation with ontologies and partly on our specific implementation based on OWL-API[11].

These services are implemented as RESTful services available via POST requests. There is also available one-step service[12] which takes ontology, transformation pattern and pattern instance on input and returns transformed ontology.

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.

2.2 Example of Ontology Transformation

In this section we provide an example of transformation pattern based on alignment pattern. We can imagine situation where in one ontology there is conceptualization of accepted paper using restriction while in other ontology it is captured as one class. This corresponds to 'Class By Attribute Value Pattern' depicted in Figure 2.

Based on this pattern transformation pattern *tp-hasValue* can be defined:

$OP1$: E={Class: A, ObjectProperty: p, Individual: a}, Ax={A EquivalentTo: (p values a)},

$OP2$: E={Class: B, Literal: An1, Literal: An2}, Ax2 ={B annotation : discr_property An1, B annotation:value An2},

PT : LI={A EquivalentTo: B, p eqAnn: An1, a eqAnn: An2}, enp(B) = make_passive_verb(a) + head_noun(A),

[7] `http://owl.vse.cz:8080/ontologyTransformation/detection/`

[8] `http://owl.vse.cz:8080/ontologyTransformation/instructions/`

[9] Ontology pattern is part of transformation pattern.

[10] `http://owl.vse.cz:8080/ontologyTransformation/transformation/`

[11] `http://owlapi.sourceforge.net/`

[12] `http://owl.vse.cz:8080/ontologyTransformation/service/`

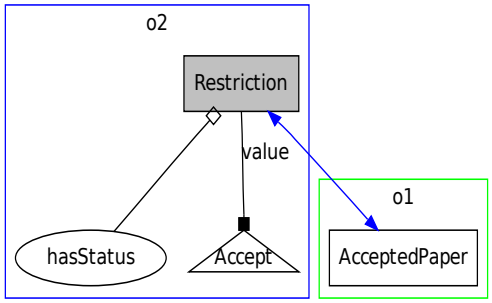

Fig. 2. Instance of alignment pattern 'Class By Attribute Value'

where the example of entity naming transformation pattern from pattern alignment enabling to make proper name for new entity, e.g. from 'Accept' as a and 'PresentedPaper' as A it makes 'AcceptedPaper' as B.

Particular instantiation of this pattern could be as it follows: $O1 : A = PresentedPaper$, $p = hasStatus$,$a = Accept$ $O2 : B = AcceptedPaper$,$An1 =' hasStatus'$, $An2 =' Accept'$, see Figure 3.

By applying tp-hasValue pattern on $O1$ we can get new entity 'AcceptedPaper' in $O1'$ which is perfectly matchable with entity 'AcceptedPaper' from $O2$ by any simple string-based technique. Besides new entity 'AcceptedPaper' there is further added an annotations which enabling reverse transformation of lost information[13]. We can use alignment pattern which is included in this transformation pattern in order to get complex correspondence, i.e.: `(O1#hasStatus value O1#Accept = O2#AcceptedPaper)`.

3 Ontology Pattern Detection

3.1 Generic variant

Generic variant of ontology pattern detection[14] does not consider final application of ontology transformation. 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. The detection of these patterns has two aspects: structural and naming ones. Our method first detect the structural aspect using the SPARQL

[13] There are different strategies how to cope with removing entities and axioms from original ontology, see [16]

[14] This is based on [12]

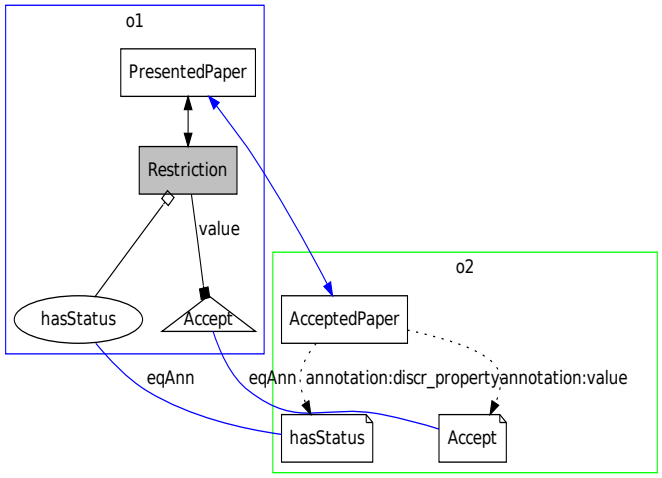

Fig. 3. Instance of transformation pattern for hasValue

language[15]. We currently use the SPARQL query engine from the Jena framework[16]. SPARQL queries corresponding to each detected pattern are detailed in sections below. Then, the method applies the lexical heuristic computing the ration between a number of sharing distinct tokens between entity ($MainEntity$) from a pattern and other entities ($Entities$) and a number of all distinct tokens of $Entities$. The particular instantiation of $MainEntity$ and $Entities$ depends on the ontology pattern, see below. This heuristic works on names of entities (a fragment of the entity URI) which are tokenised (See [13]) and lemmatized.[17] Lemmatization can potentially increase the recall of the detection process. The lexical heuristic constraint is fulfilled when the ratio is higher than 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 3.1.

Attribute Value Restriction The AVR pattern has been originally introduced in [9] 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

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

are restricted with some 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. Here, restriction class is a superclass of this named class, however we could also employ an equivalence as it was in transformation pattern, see Section 2.2. Furthermore restricting properties must be of the type 'ObjectProperty' in order to have individuals and not data types as values. Currently we do not consider the naming aspect for this pattern.

Specified Values We first considered the SV pattern in [14], but it had been originally presented in a document from the SWBPD group[18]. 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 3.1.

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.

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

Reified N-ary Relations We have already considered the N-ary pattern in [14]. It has also been an important topic of the SWBPD group [5], because there is no direct way how to express N-ary relations in OWL[19]. 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 [5]: introducing a new class for a relation which is therefore reified. For examples in the next section 3.1 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)}
```

These conditions (as one variant of detection) are not completely in correspondence with real constraint ' the reified relation class being in the range of one property'. This SPARQL query is rather one experimental way how we tried to detect this pattern. It would be worth trying other more flexible options. In order to increase the precision of the detection we also apply lexical heuristic introduced above, where 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.

Experiment In order to acquire a high number of ontologies, we applied the Watson tool[20] 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 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

[19] 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`

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

	≥ 10	(9 − 4)	3	2	1	all
AVR pattern	4	–	2	1	1	8
SV pattern	–	4	–	2	9	15
N-ary pattern	–	5	4	16	25	50

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

In the following three sections we present three detected positive examples of instances of given pattern.

AVR pattern This ontology pattern was found many times in a wine ontology[21]. One positive example is the following:

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

Chardonnay wine is restricted on these instances having value 'White' for the property hasColor. On the other hand, one negative example is the following[22]:

SV pattern The following[23] is one example which we evaluated as positive (a shared token is 'Molecule', $c = 1.0$):

$$AnorganicMolecule \sqsubseteq Molecule; OrganicMolecule \sqsubseteq Molecule$$

This can be interpreted as a collection of different kinds of molecules which is a complete partitioning. Furthermore disjointness is ensured by a query.

N-ary pattern 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[24] 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'.

Once an ontology pattern is detected, the corresponding transformation can be applied as exemplified in Section 2.2.

3.2 Specific Variant within Ontology Matching Context

Every use case imposes specific requirements on transformation which should be reflected in the phase of detection in order to increase pattern detection

[21] http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine
[22] http://sweet.jpl.nasa.gov/ontology/space.owl
[23] http://www.meteck.org/PilotPollution1.owl
[24] http://inferenceweb.stanford.edu/2004/07/iw.owl

before applying transformation. In the case of ontology matching, the detection process should consider both to-be-matched ontologies. The approach[25] will be illustrated on two tiny fragments of ontologies[26], $O1$ and $O2$. In real cases, ontologies will of course contain many more entities. ie. clustering will be more meaningful:

$O1$: $PresentedPaper \sqsubseteq Paper \sqsubseteq Document \equiv Paper \sqcap \ni hasStatus.\{Accept\}$
$O2$: $AcceptedPaper \sqsubseteq Paper \sqsubseteq Document$

We start from a set of equivalence correspondences $\langle O1 : e_i, O2 : e_j, = \rangle$ created based on an easy-to-compute lexical distance $d_L(O1 : e_i, O2 : e_j)$. Using, say, the Jaccard measure for d_L, and filtering out the correspondences below the threshold of 0.5, we get the following six correspondences:
$A = \langle O1 : Paper, O2 : Paper \rangle$, $B = \langle O1 : Paper, O2 : AcceptedPaper \rangle$,
$C = \langle O1 : PresentedPaper, O2 : AcceptedPaper \rangle$, $D = \langle O1 : Accept, O2 : AcceptedPaper \rangle$,
$E = \langle O1 : PresentedPaper, O2 : Paper \rangle$, $F = \langle O1 : Document, O2 : Document \rangle$.

Second, these correspondences are clustered based on the aggregation—here, average—of structural distances of entities (for each of $O1$,$O2$ separately) involved in them: $d(c_i, c_j) = avg(d_S(O1 : e_i, O1 : e_j), d_S(O2 : e_i, O2 : e_j))$ where c_i and c_j are correspondences between $O1$ and $O2$, and d_S is a structural distance computed as the minimal number of edges (i.e. an edge is any kind of property relating two entities) between the entities. For example, $d_S(O1 : Paper, O1 : Accept) = 2$ and $d_S(O2 : Paper, O2 : AcceptedPaper) = 1$. The initial matrix of distances between correspondences used for their clustering is in Table 2. Edge counting could of course be replaced with more elaborate ontology distance measuring, as in [2].

	A	B	C	D	E
A	-	-	-	-	-
B	0.5	-	-	-	-
C	1.0	0.5	-	-	-
D	1.5	1.0	0.5	-	-
E	0.5	1.0	0.5	1.0	-
F	1.0	1.0	2.0	2.5	1.5

Table 2.

Using hierarchical clustering we might possibly get five out of the six correspondences in the same cluster, A, B, C, D, E, in which the average distance of correspondences is 1. The output of this phase is the set of entities from $O1$ taken from this cluster: {*Paper*, *PresentedPaper*, *Accept*}. These entities would be input for the next phase, where patterns would be detected over those entities and corresponding transformation carried out.

4 Transformation Patterns vs. Alignment Patterns

As we have already mentioned *transformation patterns* consists of three components (see [16] for more details): ontology pattern A, B and pattern transformation between them. Pattern transformation consists of links between entities in order to depict which entity from ontology pattern A should be transformed to which entity from ontology pattern B. Link can be logical *equivalence correspondence* or extralogical relating two different kinds of entities, e.g. between property and class. There is further important information about how to name newly added entity or how to rename old entity.

[25] Initially presented in [15]
[26] Inspired by *OntoFarm*, `http://nb.vse.cz/~svabo/oaei2009`.

Transformation patterns can be based on matching/alignment patterns [9] considering its equivalence correspondences. But there are important differences in other aspects. From purpose perspective, while matching pattern's purpose is a representation of recurring aligning structures at the ontological level[27], purpose of transformation pattern is a representation how one structure can be transformed to conceptually similar other structure.

Furthermore, instead of a correspondences within alignment pattern there is a pattern transformation part in transformation pattern. In this part there are transformation links between entities. These links can be defined between homogeneous entities (equivalence correspondences), heterogeneous entities (*eqHet*) and between real and annotation literals (*eqAnn*). These extralogical links enable us to link logical patterns in terms of their alternatives.

Regarding transformation as such, transformation operations are defined over atomic entities - renaming, adding/removing over axioms applicable on original entities. Complex expressions are also considered within transformation pattern however in comparison with alignment pattern they are only meaningful as a part of some axiom. It does not make sense to add/remove unnamed entity (e.g. restriction class) unless it is involved in some axiom. It means that in the case of matching we consider as matchable components atomic entities and/or (even unnamed) complex expressions. On the other hand, in the case of transformation we consider as transformable components atomic entities and axioms as whole.

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*[28] (SWBPD). There are further activities in this respect like *ontology design patterns* (ODP)[29]. 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.

[27] We restrict all possible alignment patterns to them which are modeled merely at the ontological level. Alignment patterns driven by data migration are currently omitted in our transformation.

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

[29] See e.g. `http://ontologydesignpatterns.org`

In [8] the authors generally consider using SPARQL expressions for extracting Content Ontology Design Patterns from an existing reference ontology. It is followed by a manual selection of particular useful axioms towards creating new Content Ontology Design Pattern.

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[11]. 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 [7]. 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.

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 [1] 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 try to keep a clear distinction between the pattern detection and the transformation process.

Furthermore, there is a large amount of research in *ontology transformation*. It can be divided into transformation within a language (especially OWL) and across languages.

In [10] authors consider *ontology translation* from 'Model Driven Engineering' perspective[30]. They concentrate on the way how to represent the alignment as translation rules (i.e. at least in this paper on the data level). They argue that it is important to retain clarity and accessibility enable modellers to see translation problems from three aspects: semantic, lexical and syntactic. Basic shape of our transformation pattern is very similar to their metamodel. They consider having an *input pattern* which is a query and then there is an *output pattern* for creating

[30] We can expect that this work will be part of the TwoUse toolkit in future. This toolkit bridges the semantic web and model driven engineering projects, `http://west.uni-koblenz.de/twouse`

output as well as variables binding the elements. Because their approach is close to Unified Modeling Language (UML) they also base their particular text syntax of transformation rules on the Atlas Transformation Language (ATL). Input and output patterns are modelled as *MatchedRule*. Furthermore, there are variables (in patterns) and OCL expressions. However, they transfer ontology translation problem to model driven engineering. They use MOF for working with ontologies and translation rules at the data level.

In comparison with the previous work authors in [4] leverage the ontology translation problem to generic meta-model. This work has been done from *model management* perspective which implies generality of this approach. From this perspective, meta-models are *languages* for defining models. In general, model management tries to 'support the integration, evolution and matching of (data) models at the conceptual and logical design level'. In comparison with our approach there are important differences. On the one hand, they consider transformations of ontologies (expressed in OWL DL), but these transformations are considered into generic meta-model or into any other meta-model (ModelGen operator). Their approach is based on meta-model level from which each meta-model could benefit in the same way, e.g. matching, merging, transforming models between meta-models. On the contrary, in our approach we stay in one meta-model, the OWL language, and we consider transformation as a way of translating of certain representation into its alternatives.

In our current approach we base on OPPL [3] which is a macro language, based on Manchester OWL syntax, for manipulating ontologies written in OWL at the level of axioms. [16] describes how our approach uses this language.

A lot of attention has been paid to transformation between different modeling languages, transformation based on meta-modeling using UML, and specifically transformation of data-models [6].

6 Conclusion and Future Work

In this paper we have presented the workflow of the ontology transformation along with its RESTful services. Furthermore, we presented ontology pattern generic detection as well as specific variant for ontology matching context. Finally, we presented transformation patterns and their relationship with alignment patterns along with one illustrative example.

In future we should improve performance of ontology pattern generic detection as well as do more experiments with specific detection within ontology matching context. Presented specific detection approach is rather naive. In real setting, it would probably suffer from the dependency on the initial string-based matching (used as trigger for complex matching rather than 'equivalence matching' on its own). In reality, each cluster would also have to be examined for possible *inclusion of nearby entities*, e.g. by checking synonymy using a thesaurus.

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 work on extension of the ontology transformation pattern library with other ontology patterns, e.g. name patterns and other patterns based on the ODP portal.

Acknowledgements

This work has been partially supported by the CSF grant P202/10/1825 (PatOMat project). The authors would also like to thank Luigi Iannone for his support about OPPL usage.

References

1. W. Akhtar, J. Kopecky, T. Krennwallner, and A. Polleres. XSPARQL: Traveling between the XML and RDF worlds and avoiding the XSLT Pilgrimage. In *Proceedings of ESWC-08*.
2. J. David and J. Euzenat. Comparison between ontology distances (preliminary results). In *Proceedings of ISWC-08*.
3. L. Iannone, M. Egana, A. Rector, and R. Stevens. Augmenting the Expressivity of the Ontology Pre-Processor Language. In *Proceedings of OWLED-2008*.
4. D. Kensche, C. Quix, M. A. Chatti, and M. Jarke. GeRoMe: A Generic Role Based Metamodel for Model Management. In *Journal on Data Semantics*, 2007.
5. N. Noy and A. Rector. Defining n-ary relations on the semantic web, Apr. 2006.
6. B. Omelayenko and M. Klein, editors. *Knowledge Transformation for the Semantic Web*. IOS press, Amsterdam (NL), 2003.
7. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and Complexity of SPARQL. In *Proceedings of ISWC-2006*.
8. V. Presutti and A. Gangemi. Content ontology design patterns as practical building blocks for web ontologies. In *Proceedings of ER-2008*. Barcelona, Spain.
9. F. Scharffe. *Correspondence Patterns Representation*. PhD thesis, University of Innsbruck, 2009.
10. F. Silva Parreiras, S. Staab, S. Schenk, and A. Winter. Model driven specification of ontology translations. In *Proceedings of ER-2008*.
11. E. Sirin and B. Parsia. SPARQL-DL: SPARQL Query for OWL-DL. In *Proceedings of OWLED-2007*.
12. O. Šváb-Zamazal, F. Scharffe, and V. Svátek. Preliminary results of logical ontology pattern detection using sparql and lexical heuristics. In *Proceedings of WOP-2009*.
13. O. Šváb-Zamazal and V. Svátek. Analysing Ontological Structures through Name Pattern Tracking. In *Proceedings of EKAW-2008*.
14. 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.
15. O. Šváb-Zamazal, V. Svátek, J. David, and F. Scharffe. Towards Metamorphic Semantic Models. In *Poster session at ESWC-09*.
16. O. Šváb-Zamazal, V. Svátek, and L. Iannone. Pattern-Based Ontology Transformation Service as OPPL Extension. In *EKAW-2010*. To be submitted.
17. O. Šváb-Zamazal, V. Svátek, and F. Scharffe. Pattern-based Ontology Transformation Service. In *KEOD-2009*.