Adapting Ontologies to Content Patterns using Transformation Patterns

Vojtěch Svátek, Ondřej Šváb-Zamazal, and Miroslav Vacura

Department of Information and Knowledge Engineering,
University of Economics, W. Churchill Sq.4, 130 67 Prague 3, Czech Republic
{svatek|ondrej.zamazal|vacuram}@vse.cz

Abstract. Ontology content patterns are meant to be used not only for new ontologies but also for reengineering of existing ontologies. However, the modelling style of such ontologies often differs from the best-practice pattern that is to be imported to their root portions, which makes the integration of the two models time-consuming and error-prone. We explore how the recently developed PatOMat transformation framework could be applied to ease the adaptation of 'legacy' ontologies to a widely used content pattern from the OntologyDesignPatterns.org library. We also investigate the link between transformation choices and logical patterns as those earlier proposed by the W3C SWBPD Group.

1 Introduction

Ontology content patterns [4] are nowadays considered as a central artifact for promoting best practices and supporting shareability in ontology design. Although, ideally, content patterns (CPs, for brevity) should be taken into account from the very start of the design process, it is a common situation that the authors of the ontology are, at the onset, either unaware of the existence of CPs at all or too novice to choose the right one. The CPs then only enter the design process at a later phase, when at least a prototype of the ontology already exists, and their adoption has the nature of ontology reengineering.

In the easiest setting, generic CPs can be (as kind-of foundational ontology fragments) 'passed under' the current root concepts of the ontology. However, as the same conceptualisation can be expressed using different *modelling styles* in a language such as OWL,[1] the need for style transformations may often arise. A straightforward example of such transformation is change from 'class-centric' to 'property-centric' style, or vice versa, considering that the same notion can be modelled as a class (e.g. 'Purchase') or an object property ('bought_from').

In our previous work on the PatOMat project,[2] we already addressed the general need for style transformation in ontological engineering. A general *framework*, a simple transformation pattern *language*, and a set of RESTful *services*

[1] In our framework we implicitly assume the use of OWL (possibly in version OWL 2 [6]) as ontology language.
[2] http://patomat.vse.cz

(relying on external tools such as OPPL and OWL-API, see Section 2) have been developed, and thoroughly exemplified for an *ontology matching* scenario in [11]: having two ontologies to be matched, we can transform the modelling style of the one so as to make automated matching to the other easier.

The new scenario in this paper is rather one related to *ontology import*. However, compared to the generic scenario of either matching or importing an arbitrary ontology into another one, we consider here a particular, small and widely reusable ontological component—a content pattern—to which an existing 'provisional' ontology is adapted in order to gain both in rigour and comprehensibility.

The rest of the paper is structured as follows. Section 2 briefly reviews the *PatOMat* framework, transformation language and implemented prototype services, as described in [11]. Section 3 summarises the ontology matching scenario from [11], and introduces the new scenario of importing a CP into a prototype ontology. Section 4 presents the specific input setting of our case study: the *AgentRole* pattern from the *OntologyDesignPatterns.org* library, and the *ConfOf* ontology from the *OntoFarm* collection. Section 5 discusses the dimensions of ontology transformation wrt. *AgentRole*, which relate to the source pattern, target pattern, and additional axioms in the source ontology. Section 6 shows an XML serialisation of a (selected) relevant transformation pattern. Finally, Section 7 surveys some related work, and Section 8 wraps up the paper.

2 Overview of the *PatOMat* Transformation Framework

The central notion in the *PatOMat* framework[3] is that of *transformation pattern* (TP). A TP contains two *ontology patterns* (the source OP and the target OP) and the description of transformation betweem them, called pattern transformation (PT). The representation of OPs is based on the OWL 2 DL profile. However, while an OWL ontology refers to particular entities, e.g. to class *Person*, in the patterns we generally use *placeholders*. Entities are specified (i.e. placeholders are instantiated) at the time of instantiation of a pattern. An OP consists of *entity declarations*, *axioms*, and *naming detection patterns*; the last capture the naming aspect of the OP important for its detection. A PT consists of a set of *transformation links* and a set of *naming transformation patterns*. Transformation links are either *logical equivalence relationships* or *extralogical relationships* holding between two entities of different type. Naming transformation patterns serve for generating new names for old or newly created entities. Naming patterns range from *passive naming operations* such as detection of a head noun for a noun phrase to *active naming operations* such as derivation of verb form of a noun. The framework prototype implementation consists of three core services:[4]

– The *OntologyPatternDetection* service takes the transformation pattern and a particular original ontology on input, and returns the binding of entity

[3] [11] provides more details about the framework, and at `http://owl.vse.cz:8080/tutorial/` there is a fully-fledged tutorial for the current version.
[4] All accessible via the web interface at `http://owl.vse.cz:8080/`.

placeholders on output, in XML. The structural/logical aspect is captured in the structure of an automatically generated SPARQL query; the naming aspect is dealt with based on its description within the source pattern.
- The *InstructionGenerator* service takes the particular binding of placeholders and the transformation pattern on input, and returns particular transformation instructions on output, also in XML. Transformation instructions are generated according to the transformation pattern and the pattern instance.
- The *OntologyTransformation* service takes the particular transformation instructions and the particular original ontology on input, and returns the transformed ontology on output.

The third service is partly based on OPPL [2] and partly on our specific implementation over OWL-API.[5] Currently we use OPPL for the operations on *axioms* and for *adding entities*, and OWL-API for *re/naming* entities according to naming transformation patterns and for adding *OWL annotations*. As far as detection is concerned, the SELECT part of OPPL could be used to some extent; our naming constraints are however out of the scope of OPPL. Furthermore, in contrast to OPPL, we *decompose* the process of transformation into parts, which enables user intervention within the whole workflow.

3 *PatOMat* in Use: Matching vs. CP Importing Scenario

As mentioned in the Introduction, the initially considered scenario for pattern-based ontology transformation was the *ontology matching scenario*, schematically depicted in Fig. 1 (the 'structural changes' shown are merely illustrative and don't claim to correspond to meaningful transformations). The transformation step here precedes the actual matching step in the whole workflow. A given ontology (possibly just a fragment thereof), which is to be matched to a second ontology, is first matched to the source OP of a TP; the choice of the fragment as well as of the TP is however guided by an analysis of the second ontology (such that the target OP of the TP should use the same modelling style as the second ontology). The transformed ontology is built in the style of the target OP of the TP. This makes the subsequent matching to the second ontology easier; e.g. simple and fast string matching methods can be used instead of sophisticated and fragile matching methods.

The *content pattern import scenario* (adapting a prototype ontology, via transformation, to the style of a CP), depicted in Fig. 2, differs from the previous one in the overall workflow, in the sense that the import literally precedes the transformation. Namely, in the first step, the CP is included in the ontology as a separate structure, merely subordinated to *owl:Thing*. Then the transformation takes place; however, only TPs specifically tailored to the given CP are considered. In the transformed ontology,[6] shaped to the style of the target OP of the TP, the CP is already integrated into the ontology, as part of its root structure.

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

[6] For illustration, Fig. 2 also contains an entity B, which is not matched by the TP and thus implicitly copied to the transformed ontology.

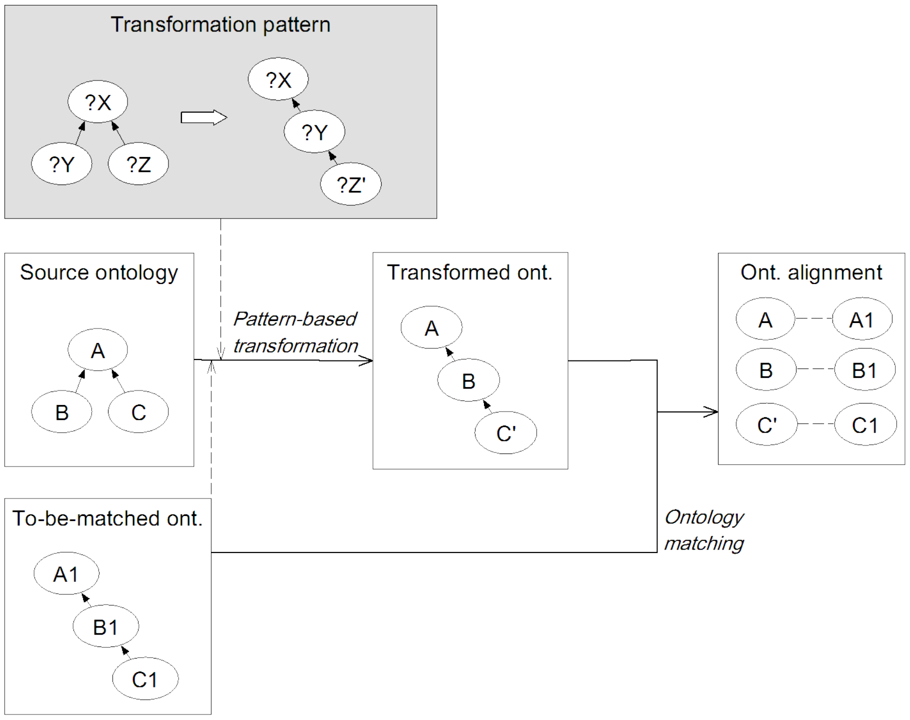

Fig. 1. Schematic depiction of transformation for ontology matching

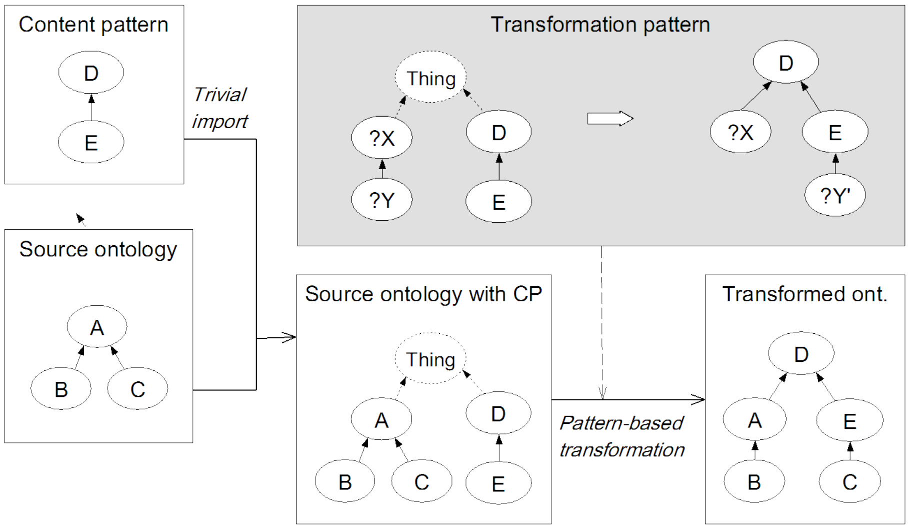

Fig. 2. Schematic depiction of transformation for CP import

Even if a part of the input to the transformation, the content pattern, is fixed, there is potentially great variability in the ways different ontologies can be adapted to the pattern. The variability goes along (at least) three different dimensions: the style of the *source pattern* occurrence proper, the style of the *target pattern* (apart from the imported content pattern itself), and the existence of *additional axioms* external to the source pattern that refer to entities from the pattern. We exemplify each of these dimensions in Section 5.

4 Input Settings for the Import Scenario Case Study

4.1 Role Modelling Approaches and *AgentRole* Pattern

The notion of *role*[7] has repeatedly appeared in the history of knowledge modelling. We will not examine this phenomenon in depth here (for thorough discussion refer e.g. to [3], Ch.7), but will only present the most obvious modelling options in the context of OWL, partially borrowed from [10].

For expressing general notions in OWL, essentially, one has two atomic entity types available. An ontology designer not deeply acquainted with the notion of ontological role will typically model a role implicitly, as

– a *class* that is a subclass of a class expressing a natural concept (e.g. *Teacher* as subclass of *Person*), such that the 'role' class is endowed with restrictions over one or more properties (e.g. an instance of *Teacher* has to be *teacherOf* some *Student* and *teacherAt* some *University*), or
– just as one or multiple separate *properties* (such as *teacherOf*, *teacherAt*) representing aspects of the role.

It should be mentioned that Sunagawa [10] also suggests a sophisticated OWL pattern for capturing all important aspects of role playing: role concepts (i.e. roles proper), natural concepts (that play roles), and role holders (that hold roles but inherit properties from natural concepts). We do not however consider this pattern here, as it is very unlikely for it to have been engineered 'just by chance' in the kind of prototype ontologies we focus on in our approach.

The wiki-based web portal at *OntologyDesignPatterns.org* contains a vast number of patterns of various types, all related to ontology design and exploitation. The most represented category is that of content patterns as immediately reusable ontology building blocks: there are 80 such patterns (though none yet certified) at the time of writing this paper. As the library aims to lower the threshold for even less experienced ontology designers, most CPs are relatively simple, yet grounded in well-designed foundational ontologies such as the lightweight version of DOLCE.[8] The *AgentRole* pattern, depicted in UML-like notation in Fig. 3, is an example of such a simple content pattern leveraging

[7] This general notion should not be confused with the term 'role' used for binary relation in description logics (as formal underpinning of OWL).
[8] `http://www.loa-cnr.it/ontologies/DUL.owl`

on the modular structure of imports. The *AgentRole* pattern, displayed with its elements in solid, specializes the *ObjectRole* pattern (entities from the 'or' namespace), which in turn specializes the *Classification* pattern (entities from the 'class' namespace), both displayed with their elements in dashed. Following from the most generic model, the *Classification* pattern merely allows to state the relationship between an entity and the concept to which this entity is somehow classified; this corresponds to an informal 'reification' of the *subClassOf* relationship. *ObjectRole*, in turn, already deals with role playing, understood as a specific type of classification;[9] entities playing roles can be any objects (i.e. no arbitrary things such as properties any more). Finally, *AgentRole* introduces an even more specific class of such role-playing objects, called *Agent*, which is declared as disjoint with class Role.

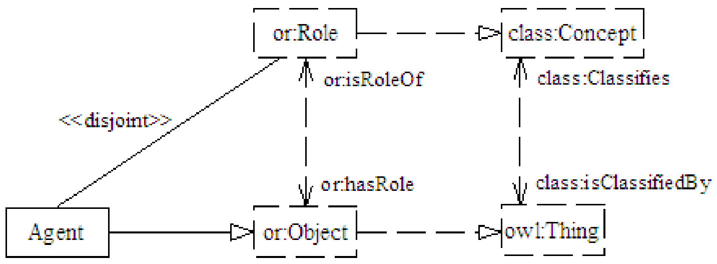

Fig. 3. *AgentRole* pattern and its imports

4.2 Example Input Ontology

As core example of input ontology to be adapted, we will use an existing ontology from the *OntoFarm*[10] collection, the *ConfOf* ontology. It models the domain of 'organizing conferences' from the point of view of a particular review (and other activity) support software: the *ConfTool*.[11] We will however also discuss patterns alternative to those used in *ConfOf*.

One 'role-playing' fragment in *ConfOf* is depicted in Fig. 4. The notion of authorship is modelled as the *Author* class being subclass of *Person*. The *Author* class has a pair of existential and universal restriction over the *writes* property, and also appears in the domain of this property and in the range of its inverse, *writtenBy*.

[9] There is a subproperty relationship (not depicted in the diagram) between *isRoleOf* and *Classifies*, and *hasRole* and *isClassifiedBy*, respectively.

[10] For an overview on the *OntoFarm* project see `http://nb.vse.cz/~svatek/ontofarm.html`.

[11] `http://www.conftool.net`

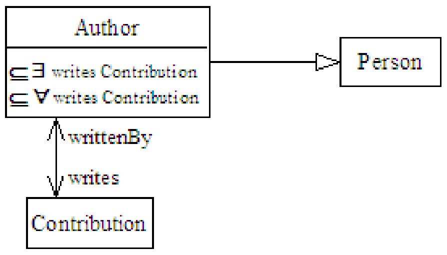

Fig. 4. Fragment of *ConfOf* ontology dealing with authorship

5 Transformation Process Alternatives

In this section we will demonstrate the aspects of the 'CP import' scenario of pattern-based ontology transformation, outlined in Section 3, on the concrete setting presented in Section 4.

5.1 Source Pattern

As natural candidates for the source pattern we can consider the simple role patterns from Section 4.1: we can call them 'class-oriented' and 'property-oriented' pattern, respectively. Obviously, *ConfOf* sticks to the 'class-oriented' role pattern. Another ontology could however, for example, avoid the Author class and model the author role merely in terms of properties such as *writes* and *written* by (or e.g. *authorOf* and *hasAuthor*), with domain/range set to *Person*, this leading to the 'property-oriented' pattern.

In addition, due to the above-mentioned sequencing of operations, the *content pattern* is already present in the ontology when the ontology is submitted to transformation, and thus has to be a part of the source pattern. However, it is unconnected to the rest of the ontology (and source pattern) yet.

5.2 Target Pattern

Obviously, the content pattern is transferred to the target pattern without change. However, the way the rest of the source pattern occurrence is shaped and linked to the content pattern may vary.

Note that, in the particular context of *AgentRole* pattern and the class-oriented role modelling style of *ConfOf*, the transformation of the notion of 'author' from a (seemingly) natural concept to a role amounts to transition from the 'instance of' relationship (being a language primitive in OWL DL) to the *hasRole* relationship (i.e. object property). By consequence, the fact of a person having the author role, which was previously expressed as e.g. "John rdf:type Author", now has to connect the individual John to some 'author role' entity through the *hasRole* property. Assuming we formalise the author role as class in the target pattern, we arrive to an instance of the "defining classes as property values" problem, treated as a logical pattern in [7].

From the set of five 'modelling approaches' (actual logical patterns, in fact) of this published pattern we will only consider those expressible in OWL DL. This eliminates "Approach 1: Use classes directly as property values". The remaining are, in turn (in the original terminology of [7]):

- Approach 2: Create special instances of the class to be used as property values
- Approach 3: Create a parallel hierarchy of instances as property values
- Approach 4: Create a special restriction in lieu of using a specific value
- Approach 5: Use classes directly as annotation property values

We will not analyze the pros and cons of different approaches wrt. particular situations, as in [7]. Instead we will try to reveal important aspects of transformation of the *ConfOf* fragment to the given approach. It should be noted that the application of the approaches is not always as obvious as in the "books-about-animals" example used through [7], as the nature of the underlying *dc:subject* property is somewhat different from the *hasRole* property in our example; we occasionally comment on such differences.

Transformation to Approach 2. It may lead, assuming some naming transformations, to the situation depicted in Fig. 5 (we omit the namespace prefixes and don't distinguish imports, for better readability). The *Author* class was removed, while there is a new class, with the same name but different meaning, subordinated to *Role*; there is now also now a distinguished instance of the latter class, called *AuthorRole*[12] (in rounded rectangle) as part of the ontology. While populating the transformed ontology, an instance of *Person* can be connected by the *hasRole* property with the *AuthorRole* individual.

ConfOf models authorship in a simple way such that there are no subclasses of *Author*. However, if there were such subclasses (e.g. *PosterAuthor*), they would be transformed to subclasses of the new *Author* class; each such class would also have a corresponding individual (e.g. *PosterAuthorRole*) as *direct* instance.

Transformation to Approach 3. The transformation corresponds to the setting in Fig. 6. The difference from Approach 2 is that *Author* has no longer the semantics of role and thus is not subclass of *Role*. As *AuthorRole* is still instance of *Role*, we link it to *Author* using the annotation property *rdfs:seeAlso*.

The most important distinction is however not visible in this simple diagram. Namely, if there were a subclass system under *Author* in the original ontology, it would have to be transferred to the instance level. Instead of subclasses, there

[12] In the "books-about-animals" example in [7], the class is assumed to correspond to an animal species and the instance to the 'topic' of this animal. Both in the original example and in the example used in this paper, the nature of such instances—'roles' and 'topics', respectively—is thus not very coherent with the semantics of the class (whose name, be it e.g. *Lion*, or *Author*, is rather appropriate for a natural concept).

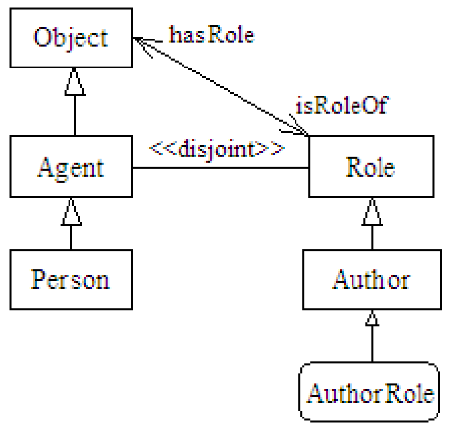

Fig. 5. Target of transformation using Approach 2

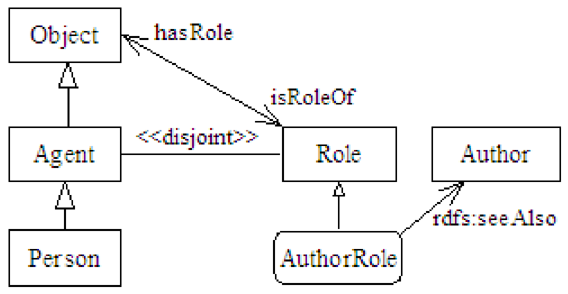

Fig. 6. Target of transformation using Approach 3

would be individuals such as *PosterAuthorRole*, which would be all direct instances of *Role*, and their specialization relationship would be modelled by a dedicated object property such as *subRoleOf*.

Transformation to Approach 4. This is similar to Approach 2 (including the treatment of a hypothetical subclass system), except that

- the original *Author* class is not removed
- consequently, the new class (subclass of *Role*) is not named *Author* but *AuthorRole*
- no special instances are generated for the new (*AuthorRole*) class
- instead, an additional restriction (in bold) is imposed on the *Author* class, relating it to the *AuthorRole* class.

The result is in Fig. 7.

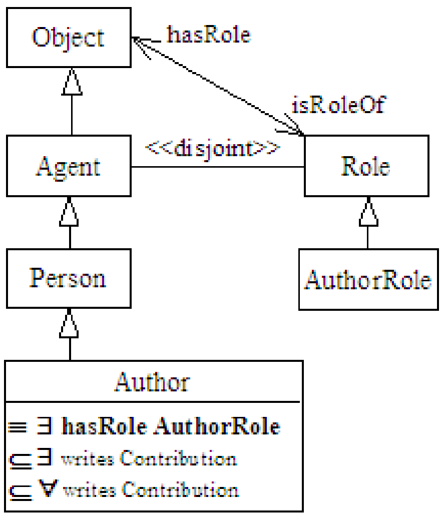

Fig. 7. Target of transformation using Approach 4

Transformation to Approach 5. The transformation corresponds to the setting in Fig. 8. In this approach, similarly to Approach 3, *Author* is not subclass of *Role*, and represents, together with its hypothetical subclasses, a branch of the ontology separate from *Person*, too. The *Role* class and the original *hasRole* and *isRoleOf* object properties, although imported with the *AgentRole* pattern, are not used at all. Instead, a new pair of *annotation properties*, borrowing the name of these two object properties, is defined. They can be used to directly connect instances of *Person* to class *Author* (or its subclass), as using classes as values of annotation properties is possible within OWL DL. Obviously, the downside is unavailability of information in annotations to conventional DL reasoners.

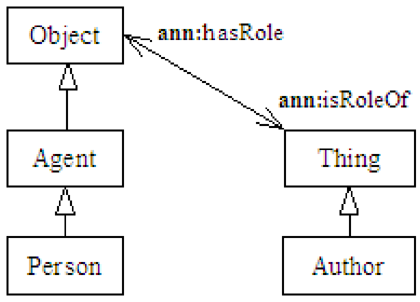

Fig. 8. Target of transformation using Approach 5

Feedback to the W3C Pattern. The context of transformation probably makes obvious that the five approaches from [7] (including Approach 1, which amounts to using a class directly as value of the property, thus lifting the ontology to the OWL Full dialect) are not the only possible choices for expressing the given conceptualization. Systematic variation of the (declaratively expressed) transformation pattern can give rise to multiple new approaches, which deserve to be further systematized, in a new round of the best practice identification process.

5.3 Additional Axioms

The implemented version of the transformation framework does not make distinction between the source pattern proper (i.e. structures whose occurrence is essential for the presence of the pattern) and additional, 'external' axioms that only 'touch' the pattern by referring to one of its entities. Such axioms may or may not be present for an ontology fragment to match the source pattern, but if they are present then they have to be considered by the transformation. Considering them as mandatory makes the whole transformation pattern over-specific and hard to understand. Therefore, for the new (not yet implemented) version of the transformation framework, we will decompose the transformation patterns into a mandatory part (containing the source and target ontology patterns proper, and their pattern transformation) and an optional part (containing the additional axioms, and their pattern transformation).

Specifically, when looking at Fig. 4, we see that the original *Author* class is used both in local (existential and universal) and global (domain and range) restrictions. If we use e.g. Approach 2, the *Author* class is however removed. The easiest but most lossy way of dealing with axioms that referred to it would be to drop the local restrictions together with the class, and to let the global restrictions refer to the immediate superclass (i.e., *Person*). However, we can also (at the cost of complexity overhead) replace the removed named class with the composed class expression $Person \sqcap \exists\ hasRole.Author$ in these axioms. While this is straightforward for the global restrictions (just setting the domain/range of the respective properties to be this class expression), for modelling the local restrictions in OWL 2 DL we would have to employ an explicit anonymous class in order to link the two composed concept expressions, e.g.:

$$\_1 \equiv Person \sqcap \ \exists\ hasRole.Author$$
$$\_1 \sqsubseteq \exists\ writes.Contribution$$

Detailed discussion of the impact of such manipulations is however beyond the scope of this paper.

6 XML Serialisation of Transformation Patterns

Fig. 9 shows the XML serialisation of the transformation pattern for Approach 2. The codes for all four mentioned approaches are available in the transformation pattern library accessible from `http://nb.vse.cz/~svabo/patomat/tp/`.

As mentioned in Section 2, the transformation pattern consists of two ontology patterns, the source and the target one, plus a pattern transformation. Both OPs contain the given CP (*AgentRole*), with concrete classes, in their 'axioms' part (last six axioms in each OP).

In addition, the *source OP* declares two placeholders for classes, ?A and ?B, such that the second is subclass of the first (such as *Author* is of *Person*): this is the only axiom with placeholders. There is no naming detection pattern, for simplicity (as none is needed in this simple example). Additional axioms, mentioned in Section 5.3, are not yet considered either.

The *target pattern* declares two placeholders for classes, ?C and ?D, and one for individual, ?b, which is instance of the latter class. The first two of its axioms however already interlink the imported CP with placeholder classes (this corresponds to subordinating *Person* to *Agent* and *Author* to *Role*). The third axiom states that ?b is instance of ?D (such as *AuthorRole* to *Author*).

The *pattern transformation*, finally, declares the logical equivalence[13] transformation links between the classes from the source and target OP, and a simple naming pattern transformation for creating the name of the new individual by concatenating the name of class ?B with the string 'Role'.

7 Related Work

We are unaware of any style transformation approach used in connection with *content patterns* as in our current research. As regards our framework as such, several generic approaches to *ontology transformation* have recently been published. We refer here to two that look most relevant to our work (aside pure OPPL, which we ourselves use as an external component of our framework).

In [9] the authors consider *ontology translation* from the Model Driven Engineering perspective. The basic shape of our transformation pattern is very

[13] For simplicity we use an equivalence link even for ?B vs. ?D, although the equivalence between the class *Person* before and after transformation is arguable; removing a class and creating a new one would be sounder.

```
<tp>
  <op1>
    <entity_declarations>
      <placeholder type="Class">?A</placeholder>
      <placeholder type="Class">?B</placeholder>
    </entity_declarations>
    <axioms>
      <axiom>?B subClassOf ?A</axiom>
      <axiom>or:isRoleOf range or:Object</axiom>
      <axiom>or:hasRole domain or:Object</axiom>
      <axiom>or:hasRole range or:Role</axiom>
      <axiom>or:isRoleOf domain or:Role</axiom>
      <axiom>:Agent subClassOf or:Object</axiom>
      <axiom>:Agent disjointWith or:Role</axiom>
    </axioms>
  </op1>
  <op2>
    <entity_declarations>
      <placeholder type="Class">?C</placeholder>
      <placeholder type="Class">?D</placeholder>
      <placeholder type="Individual">?b</placeholder>
    </entity_declarations>
    <axioms>
      <axiom>?C subClassOf :Agent</axiom>
      <axiom>?D subClassOf :Role</axiom>
      <axiom>?b a ?D</axiom>
      <axiom>or:isRoleOf range or:Object</axiom>
      <axiom>or:hasRole domain or:Object</axiom>
      <axiom>or:hasRole range or:Role</axiom>
      <axiom>or:isRoleOf domain or:Role</axiom>
      <axiom>:Agent subClassOf or:Object</axiom>
      <axiom>:Agent disjointWith or:Role</axiom>
    </axioms>
  </op2>
  <pt>
    <eq op1="?A" op2="?C"/>
    <eq op1="?B" op2="?D" />
    <ntp entity="?b">?B+Role</ntp>
  </pt>
</tp>
```

Fig. 9. Simple transformation pattern for Approach 2

similar to their metamodel. They consider an *input pattern*, i.e. a query, an *output pattern* for creating the output, as well as variables binding the elements. However, the transformation is considered at the *data level* rather than at the *schema level* as (primarily) in our approach.

In comparison with the previous work the authors of [5] leverage the ontology translation problem to the generic meta-model. This work has been done from the *model management* perspective, which implies a generality of this approach. There are important differences to our approach. Although they consider transformations of ontologies (expressed in OWL DL), these transformations are directed into the generic meta-model or into any other meta-model such as that of UML or XML Schema. In contrast, in our approach we stay within one meta-model, the OWL language, and we consider transformation as a way of translating a certain representation into its modelling alternatives.

8 Conclusions and Future Work

We demonstrated that transformation patterns are a useful mediator for adequately importing content patterns into ontologies, especially into prototype ones that have been designed ad hoc, are not yet widely used, and now are to be put (through the content patterns) onto a more solid and shared foundation. Although the scenario used in this paper is quite narrow, we believe that it uncovers recurrent issues related to ontology style heterogeneity in general.

Aside the *AgentRole* pattern and its generalizations, we plan to explore other *content patterns* from *OntologyDesignPatterns.org*. Analogously, we will experiment with different *input ontologies*, which will require new transformation operations such as changing between properties and classes ('de/objectification'), as e.g. in the 'n-ary relations' logical pattern [8]. Obviously, the *cost/benefit ratio* of the CP import functionality should also eventually be examined with respect to the size of the ontology to be transformed and amount of data that already refer to it (and have to be transformed too, either at query time or in bulk).

The most critical future work, which is not specific for content pattern import but generic for pattern-based transformation, is however support for *recursive* and *optional* parts of patterns. This will lead to much more efficient transformation, as the transformation process, triggered at a 'root' entity (such as the *Author* class in *ConfOf*) could be propagated down the subclass (for different specific types of authors) or subproperty links, and yield an analogous, though stylewise different structure in the target ontology.

Another generic functionality we also plan to achieve in long term is systematic, pattern-based swapping of the information lost during style transformation into *OWL 2 annotations*, see e.g. [1] for initial considerations.

From the research point of view, a challenging task is to *automatically suggest* fragments of ontologies that should be (transformed and) subordinated to a content pattern. For the *AgentRole* pattern, for example, this challenge amounts to recognising classes or properties that implicitly express a role. In [12] we already formulated and in [13] made an initial evaluation (with promising result) of a heuristic for detection of a 'role' class, through the occurrence of its name in (a naming pattern context of) a property having this class as domain. Much more complex heuristics, possibly inductively learnt, would however be needed for efficient recommendation.

This research has been partially supported by the CSF grant no. P202/10/1825, "PatOMat – Automation of Ontology Pattern Detection and Exploitation".
The authors are indebted to Eva Blomqvist, Aldo Gangemi and Valentina Presutti for inspiring discussions and hints regarding the ODP.org content patterns.

References

1. Annotation System. OWL WG, Work-in-Progress document, `http://www.w3.org/2007/OWL/wiki/Annotation_System`.
2. Egaña M., Stevens R., Antezana E.: Transforming the Axiomisation of Ontologies: The Ontology Pre-Processor Language. In: OWLED. 2008. (z related work:)
3. Guizzardi G.: *Ontological Foundations for Structural Conceptual Models*, PhD Thesis, University of Twente, The Netherlands. Published as the book Ontological Foundations for Structural Conceptual Models, Telematica Instituut Fundamental Research Series No. 15, ISBN 90-75176-81-3 ISSN 1388-1795; No. 015; CTIT PhD-thesis, ISSN 1381-3617; No. 05-74.
4. Presutti V., Gangemi A.: Content ontology design patterns as practical building blocks for web ontologies.: In Proceedings of ER2008. Barcelona, Spain, 2008.
5. Kensche D., Quix C., Chatti M., Jarke M.: GeRoMe: A Generic Role Based Metamodel for Model Management. In: *Journal on Data Semantics*, Vol.8, p.82–117, 2007.
6. Motik B., Patel-Schneider P.F., Parsia B. (eds.): OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax. W3C Recommendation 27 October 2009, online at `http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/`.
7. Noy N. (ed.): Representing Classes As Property Values on the Semantic Web. W3C Working Group Note 5 April 2005, online at `http://www.w3.org/TR/swbp-classes-as-values/`.
8. Noy N., Rector A. (eds.): Defining N-ary Relations on the Semantic Web. W3C Working Group Note 12 April 2006, online at `http://www.w3.org/TR/swbp-n-aryRelations/`.
9. Parreiras F. S., Staab S., Schenk S., Winter A.: Model Driven Specification of Ontology Translations. In: 27th International Conference on Conceptual Modeling (ER-2008). n. 5231, p. 484–497. 2008.
10. Sunagawa E., Kozaki K., Kitamura Y., Mizoguchi R.: Role organization model in Hozo. In: Proc. EKAW'06, Podebrady, Czech Republic. Springer, LNCS.
11. Šváb-Zamazal O., Svátek V., Iannone L.: Pattern-Based Ontology Transformation Service Exploiting OPPL and OWL-API. In: EKAW-2010, Lisbon, Portugal, 2010.
12. Svátek V., Šváb-Zamazal O., Presutti V.: Ontology Naming Pattern Sauce for (Human and Computer) Gourmets. In: Workshop on Ontology Patterns at ISWC'09, Washington DC, 2009. Online `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-516/`
13. Svátek V., Šváb-Zamazal O.: Entity Naming in Semantic Web Ontologies: Design Patterns and Empirical Observations. In: Znalosti 2010, 9^{th}Czecho-Slovak Annual Knowledge Technology Conference, Jindřichův Hradec 2010, Czech Republic.