Constructs Replacing and Complexity Downgrading via a Generic OWL Ontology Transformation Framework

Ondřej Šváb-Zamazal[1], Anne Schlicht[2], Heiner Stuckenschmidt[2], and Vojtěch Svátek[1]

[1]University of Economics, Prague, {ondrej.zamazal, svatek}@vse.cz
[2]University of Mannheim, {anne, heiner}@informatik.uni-mannheim.de

Abstract. Many of the tools supporting the OWL ontological language face complexity problems when handling certain constructs of the language. This leads to the requirement of automatically changing the ontology, either by removing a specific type of construct or by adhering (downgrading) the ontology to a predefined OWL2 profile such as OWL2 EL. We present an approach to construct replacing and complexity downgrading that relies on transformation patterns processed by a generic ontology transformation framework. Transformation patterns allow to declaratively formulate and transparently execute axiom replacement operations. This potentially preserves derivations that would otherwise be lost due to simple removal of problematic axioms.

1 Introduction

Existing tools operating on ontologies normally support a certain, well defined, set of logical operators. In many cases this set of operators is not sufficient to completely capture the semantics of the OWL language. As a result, these tools cannot be used on certain ontologies or they provide incomplete reasoning results. In both cases, the transformation of the input ontology can improve the situation. In particular, the ontology can be transformed into a version that only uses the supported operators. Doing this outside the tools gives the user more flexibility because (s)he can design a transformation that is not directly hard-coded into the tool.

In our previous work on the PatOMat project[1] we already addressed the general need for 'style' transformation in ontological engineering. In this paper we are concerned with the 'language profiling' scenario of transformation, i.e. replacing certain OWL constructs that could be hard for some tools. This replacement task is supported by a general *ontology transformation framework* [12], a simple *transformation pattern language*, and a set of web-based *services* relying on external tools such as the Ontology Pre-Processor Language (OPPL) and OWL-API, see Sec. 2. The current paper extends [12] with a description of the

[1] `http://patomat.vse.cz/`

language profiling scenario with its two use cases, a larger collection of transformation patterns, an experiment, new features of the core implementation and a new web-based application.

The rest of the paper is structured as follows. Sec. 2 briefly reviews the *PatOMat* framework, transformation language and processing services, in the current form. Sec. 3 introduces the language profiling scenario with its pipeline. This scenario is then split into two use cases; the first one (Sec. 4) describes the 'on purpose' construct replacement use case, while the second (Sec. 5) deals with complexity downgrading. Complexity downgrading is an extension of the first scenario in terms of applying more than one transformation pattern dynamically composed into a sequence according to recommendations from ontology analysis. Furthermore, an experiment is presented in Sec. 5.2 that deals with the second use case. Finally, Sec. 6 surveys related work, and Sec. 7 discusses the benefits of the approach and wraps up the paper.

2 *PatOMat* Transformation Framework

The central notion in the *PatOMat* framework[2] is that of *transformation pattern* (TP). A TP contains two *ontology patterns* (source OP and target OP) and the description of the transformation betweem them, called *pattern transformation* (PT). For instance, we can specify a TP such that a subsumption relation (as source, OP1) should be transformed to a SKOS[3] taxonomic relationship (as target, OP2). A schematic description follows.

```
OP1: ?OP1_A subClassOf ?OP1_B
OP2: ?OP2_A skos:broader ?OP2_B
PT:  ?OP1_A~?OP2_A    ?OP1_B~?OP2_B.
```

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*, e.g. $?OP1\_A$. Entities are specified (i.e. placeholders are instantiated) at the time of instantiation of a pattern. An OP consists of *entity declarations* (referring to placeholders or concrete entities), *axioms* and *naming detection patterns*; the last capture the naming aspect of the OP important for its detection.[4] 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 pairs of entities of different type (such as class vs. individual, as in our example above). 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 a verb form of a noun.

[2] [12] provides more details about the framework, and at `http://owl.vse.cz:8080/tutorial/` there is a fully-fledged tutorial for the current version.

[3] `http://www.w3.org/TR/skos-primer/`

[4] The naming aspect is less important for language profiling than it is for ontology matching or importing (merging).

For instance, the abovementioned TP would transform the following OWL ontology fragment

```
Paper subClassOf Document. Review subClassOf Document.
ConferencePaper subClassOf Paper. JournalPaper subClassOf Paper.
```

to the SKOS terminology fragment

```
Paper skos:broader Document. Review skos:broader Document.
ConferencePaper skos:broader Paper. JournalPaper skos:broader Paper.
```

The framework prototype implementation is available either as a *java library* or as three *core services*.[5] The java library is directly used in a web-based application briefly described in Sec. 5. The whole transformation is divided into three steps, which correspond to the three services:

- The *OntologyPatternDetection* service takes the transformation pattern and a particular original ontology on input, and returns the binding of entity placeholders on output, in XML. The structural/logical aspect is captured in the structure of an automatically generated SPARQL query;[6] 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.[7] 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.

The framework has been recently enriched with several advanced features such as *recursive* processing of structures in ontologies in a single step both in the detection phase and in the actual transformation phase. Furthermore, multiple alternative strategies can be applied in handling *additional axioms*, i.e. axioms that are not a literal part of an input pattern but get affected by its transformation; in this case removal can be allowed for both axioms and entities (with additional options that we omit for brevity), axioms only, or none.

[5] All accessible via the web interface at `http://owl.vse.cz:8080/`.
[6] `http://www.w3.org/TR/rdf-sparql-query/`
[7] `http://owlapi.sourceforge.net/`

The framework was previously explored for two other scenarios. The *ontology matching* scenario, where two ontologies are to be matched, is based on the idea that we can transform the modelling style of one ontology so as to make automated matching to the other ontology easier [12]. Another scenario deals with importing (merging) a best-practice *ontology content pattern* into an existing 'provisional' ontology, which thus needs to be adequately adapted [11].

3 *PatOMat* in Use: Language Profiling Scenario

In comparison with those other scenarios, the language profiling scenario leads to a fully automatic pipeline. First, a source ontology is pre-processed in order to syntactically decompose the constructs that can hinder querying in a unified way. In our case we decompose ($\rightarrow$) the following constructs:

- *intersection*: A subClassOf (B and C) $\rightarrow$ A subClassOf B. A subClassOf C.[8]
- *disjointness*: DisjointClasses(B, C, D) $\rightarrow$
  B disjointWith C. C disjointWith D. B disjointWith D.
- and *disjoint union*: DisjointUnion(B, C, D)[9] $\rightarrow$
  B equivalentTo C or D. C disjointWith D.

The next step is a detection performed by the *OntologyPatternDetection* service. There is typically more than one pattern instance as a result of the detection step. Furthermore, it is usually precise, because in the case of the language profiling scenario, detection is merely based on structural/logical aspects and naming detection patterns are mostly not needed. The following step amounts to generation of transformation instructions by the *InstructionGenerator* service. Finally, the application of instructions is carried out by the *OntologyTransformation* service according to the selected transformation strategy. By default, it uses the 'progressive' transformation strategy, which enables the removal of axioms but not the removal of entities.

This is the basic pipeline of the language profiling scenario as applied in the first use case (cf. Sec. 4). In contrast, the complexity downgrading use case (cf. Sec. 5) slightly modifies the pipeline by adding an analysis of the source ontology to specify which transformation patterns should be applied. Consequently, selected transformation patterns are dynamically composed into a sequence. Finally, a post-processing step is performed for ensuring completness of the process.

The transformation of language profiling constructs can be generally done in three ways: either they can be replaced with an *equivalent different representation*, or they can be replaced with an *approximate different representation*, or they can be *removed*. The first option is obviously the best one. However, it is only rarely possible to find an equivalent representation using other constructs

[8] For writing axioms we use the intuitive Manchester syntax available at: `http://www.w3.org/TR/owl2-manchester-syntax/`

[9] B is the disjoint union of C and D.

when in need to eliminate a problematic construct during complexity downgrading. The second option is more realistic. However, there is often no (even approximate) alternative way of representation and the problematic construct has to be simply removed.

4 Ontology Transformation for Specific Language Construct Replacement

Transformation can be driven by a request for replacing a specific language construct. In this section we provide an example dealing with nominals. Tackling nominals can be problematic for some reasoners. In the following example we will show how nominals can be replaced rather than removed. Let us assume that we have nominals describing continents and the `Continent` class defined as 'one of' those nominals (implicitly assuming their mutual difference):

```
Continent equivalentTo {Africa, America, Antarctica, Asia, Australia, Europe}.
```

Let us further assume that we have the `AfricanRedSlip` class,[10] defined via the `hasContinentOfOrigin` property:

```
AfricanRedSlip subClassOf Ware.
AfricanRedSlip subClassOf (hasContinentOfOrigin value Africa).
```

Nominals could be simply removed; however then we would lose e.g. part of the description of `AfricanRedSlip`. Instead, we can replace a set of nominals by union of helper classes *xxx*`_nc` each one accomodating exactly one original instance of the nominal class:

```
OneOfContinent equivalentTo (Africa_nc or America_nc or
        Antarctica_nc or Asia_nc or Australia_nc or Europe_nc).
Africa a Africa_nc. America a America_nc.
Antarctica a Antarctica_nc. Asia a Asia_nc.
Australia a Australia_nc. Europe a Europe_nc.
```

This transformation is approximate because it is no longer assured that e.g. `Africa_nc` could not have other individuals than Africa. Due to this change we should also modify the description of `AfricanRedSlip`:

```
AfricanRedSlip subClassOf Ware.
AfricanRedSlip subClassOf (hasContinentOfOrigin some Africa_nc).
```

This can be done automatically using our framework with a specific TP.[11] It is worth noting that, historically, nominals used to be represented in this ('transformed') way.

[10] African red slip is a kind of ancient pottery, see `http://open.vocab.org/docs/AfricanRedSlip`.

[11] The pattern is available from `http://nb.vse.cz/~svabo/patomat/tp/lr/tp_nominals-6a.xml`

Further off-the-shelf transformation patterns for replacing different OWL constructs are available online.[12] They can be divided into three groups: equivalent, approximate and removed representations (cf. Sec. 3).

5 Ontology Transformation for Complexity Downgrading of an Ontology

The transformation can also be driven by an ontology complexity requirement of some tool. In such a case, transformation usually comprises more than one transformation pattern in order to achieve the required complexity level. In this work we focus on the complexity level corresponding to the OWL2EL profile [7], since this profile is supported by many tools, e.g. the ELOG-reasoner [8]. In comparison with the use case from the previous section there are two more steps. Based on a given list of forbidden constructs, *ontology analysis* figures out (by using the OWL-API library) which transformation patterns have to be executed. These patterns are added into a sequence of transformation patterns and then executed by the transformation in a sequential order. Additionally there is a *post-processing* step where the remaining forbidden constructs are removed using the OWL-API library. This step ensures completeness of the downgrading process.

Both use cases from Sec. 4 and 5 are supported by a *web-based application*.[13] Following the input of the source ontology URI (and selected TP in the first use case), the transformed ontology is displayed (together with a brief transformation log) and a link to its code is also provided.

5.1 Transformation Patterns Employed in Downgrading to OWL2EL Profile

There are several OWL 2 constructs that are not supported in the OWL2EL profile [7]: universal quantifications to a class expression, cardinality restrictions, disjunctions, class negations, enumerations involving more than one individual, disjoint properties, irreflexive object properties, inverse object properties, functional and inverse-functional object properties, (a)symmetric object properties.

It is generally difficult to find some replacement of unsupported constructs since their replacement usually leads to using other unsupported constructs, e.g. ObjectMaxCardinality could be replaced by a complemented ObjectMinCardinality restriction, which is equally forbidden in OWL2EL.

In the following, we go through three different language constructs that can be replaced using our transformation patterns (replacement transformation). We describe them briefly and exemplify the preserved implications. However, let us

[12] `http://nb.vse.cz/~svabo/patomat/tp/lr/`; there is a link to the XML serialization of each pattern, a short description, and an ontology on which the pattern can be tested.

[13] Available from `http://owl.vse.cz:8080/Downgrading/`.

note that different solution as a transformation pattern can be suggested and applied in the framework. Finally, we provide an experiment illustrating the effect of transformation on query answering results.

Complement of Universal Restriction The complement construct is not allowed in OWL2EL at all. We can approximately replace it using existential restriction wrt. the top concept, i.e. instead of having

```
PizzaWithTopping subClassOf (not (hasTopping only Tomato))
```

we will have the following

```
PizzaWithTopping subClassOf (hasTopping some Thing)
```

In order to exemplify the preservation of derivations, let us consider that we also have the following axioms in our TBox:

```
(hasPizzaIngredient some Thing) subClassOf Pizza
hasTopping subPropertyOf hasPizzaIngredient
```

If the problematic axiom were only removed and not replaced the following subsumption could not be inferred:

```
PizzaWithTopping subClassOf Pizza
```

The corresponding transformation patterns described in this paper are available online.[14]

Minimum Cardinality Cardinality restrictions are not allowed in OWL2EL. Minimum cardinality of 1 can be *equivalently* replaced by an existential restriction applied on the same filler class. In the case that the minimum cardinality is higher than one, an *approximate* transformation can be applied.

For example, instead of having

```
AcceptedPaper subClassOf (hasDecision min 2 Acceptance)
```

we will have the following

```
AcceptedPaper subClassOf (hasDecision some Acceptance)
```

In order to exemplify the preservation of derivations, let us consider that we also have the following axioms in our TBox:

```
EvaluatedPaper = hasDecision some Decision
Acceptance subClassOf Decision
```

This implies

```
AcceptedPaper subClassOf EvaluatedPaper
```

[14] `http://nb.vse.cz/~svabo/SOFSEM2013/`

Enumerations of More than One Individual The OWL2EL profile only permits enumeration of one individual, therefore transformation must be carried out in the cases with higher number of individuals. We suggest the following approximate transformation. Instead of having

```
EurAsia = {europe, asia}
```

we will have the following

```
Europe_nc = { europe }. Asia_nc = { asia }.
Europe_nc subClassOf EurAsia
Asia_nc subClassOf EurAsia
```

We assume that the individuals `Europe` and `Asia` are different. However, in this way we cannot express that every `EurAsia` is either `Europe_nc` or `Asia_nc`.

In order to exemplify the preservation of derivations, let us consider that we also have the following axioms in our TBox:

```
EuropeanWatch = ( hasContinentOfOrigin hasValue europe )
EurAsiaWatch = ( hasContinentOfOrigin some EurAsia )
```

This implies

```
EuropeanWatch subClassOf EurAsiaWatch
```

5.2 Experiment

We performed an experiment about the effects of *replacement transformation* in comparison with *removal transformation*, which simply removes the axioms involving forbidden constructs by OWL-API. The experiment had three steps:

1. *Ontology collection gathering.* In order to gather collection of ontologies we used the Watson semantic search.[15] We applied four selection criteria for selecting ontologies into the collection: OWL ontology language (target language of the framework), more than 10 classes, more than 5 properties (ontologies should not be too small), and absence of imports (current limitation of the OPPL tool and thus of the framework). This gives us 328 ontologies. Final criterion says that an ontology must have at least one forbidden construct transformable by transformation patterns. This reduced the set of ontologies to 63. However, due to parsing problems (in OWL-API or Jena[16]), other syntactical problems in ontologies and inconsistent ontologies, we had finally 38 ontologies in our experimental collection.
2. *Transformation of ontologies.* We transformed each of these original ontologies (O variant of an ontology; see in the `ontologies` directory[17]) into the OWL2EL profile using *removal transformation* (R variant of an ontology; see in the `ontologiesR` directory), using simple modifications of ontologies such as adding declarations of classes and properties using OWL-API (ST variant of an ontology; see in the `ontologiesST` directory) and using *replacement transformation* with an application of our transformation patterns (T variant of an ontology; see in the `ontologiesT` directory).

[15] `http://kmi-web05.open.ac.uk:8080/WatsonWUI/`

[16] `http://jena.apache.org/`

[17] Detail web-page report about the experiment along with downloadable collection of ontologies is at: `http://nb.vse.cz/~svabo/SOFSEM2013/`

3. *Comparison of number of preserved subsumption relations.* Finally, for each transformed version of an ontology we computed the subsumption relations included in the ontology explicitly (using ARQ in Jena) or implicitly (using ARQ in Jena and Pellet reasoner[18]). The generated query was in the following shape: *ASK Class1 rdfs:subClassOf Class2* .[19] Then we automatically compared the preserved subsumption relations in the R variant wrt. subsumption relations in the original ontology, preserved subsumption relations in the ST variant wrt. subsumption relations in the original ontology, and, finally, preserved subsumption relations in the T variant wrt. subsumption relations in the original ontology.

The number of problematic axioms (obtained using OWL-API) ranged from 11 to 2133. Besides the forbidden constructs listed in Sec. 5.1 there were forbidden datatypes in data range and undeclared classes or properties.[20] The number of minimum cardinality replacement transformations ranged from 1 to 120 applied on all ontologies in the collection and the number of enumeration replacement transformations ranged from 1 to 27 only applied on 8 analysed ontologies.

In total, there were 17 ontologies in which removal transformation had no negative effect on subsumption relations (Table 1), including 3 ontologies which had no subsumption relation in the original ontology at all. Next, for 13 ontologies any kind of transformation did not save subsumption relations. It turns out that simple modifications (ST variant) improved 7 ontologies with regard to lost subsumption relations ranging from 1 to 75 saves. Detailed analysis showed that those seven ontologies missed classes or properties declarations and these were simply added using OWL-API. Without these modifications the removal transformation by OWL-API simply removed all axioms in which the problematic entities were involved. Consequently, this also removed asserted subsumption relations. Finally, there was only one positive effect caused by minimum cardinality replacement transformation, in which the number of missing subsumption relations decreased from 74 to 62.

Let us have a closer look at one example of preserved subsumptions there. The replacement transformation preserved, for instance, the following subsumption relations (an equivalence is decomposed into (1) and (2)):

```
(1) Module subClassOf StructuralElement
(2) StructuralElement subClassOf Module
```

These subsumption relations are derived based on the following axioms:

```
Module subClassOf (element min 1).
StructuralElement subClassOf (element min 1).
element Domain Module.
element Domain StructuralElement.
```

[18] `http://clarkparsia.com/pellet`

[19] Class1 and Class2 were iteratively bound with all combinations of named classes from given ontology.

[20] Although a declaration is not matter of logic, an OWL ontology without declarations is incomplete and thus in the OWL Full profile.

Thanks to replacement transformation in which "Module subClassOf (element min 1)" was replaced by "Module subClassOf (element some Thing)" (analogically for StructuralElement) the (1) and (2) relations were preserved.

This weak overall effect (potentially even intensified considering the 265 ontologies in which no replaceable forbidden constructs were identified[21]) can be explained by the fact that the current replacement transformation patterns cover a small set of all problematic issues only. However, any newly designed transformation pattern can be employed within the process in the future. Next, the replacement trasformation can only have a positive impact (in the setting of our experiment) if there are further axioms due to which subsumption relations can be derived (as demonstrated step-by-step for each replacement transformation in Sec. 5.1). Last but not least, we should also consider that this experiment only evaluates the effect of preserved subsumption relations but it does not evaluate the first mentioned use case, which is an "on purpose" construct replacement use case (Sec. 4). This should be accordingly reflected in a future experiment.

	number of ontologies
no difference between O and R variants	17
no positive effect wrt. saved subsumption relations	13
saved subsumptions due to simple modifications	7
saved subsumptions due to replacement transformations	1

Table 1. Effects summary

Regarding the time performance, which includes pattern detection, instructions generation and transformation, a cardinality replacement transformation takes approximately ten seconds, while an enumeration replacement transformation takes twenty seconds. In the case of enumeration replacement transformation the time increases with a number of transformations because it is applied iteratively over an ontology, while a cardinality replacement transformation runs only once for all applicable cardinality transformations in an ontology.

6 Related Work

Prior research on ontology simplification can be divided into generic approaches and those specifically tailored for a certain (popular) reasoner. Additionally we also consider general approaches to ontology transformation (not confined to simplification). The following three paragraphs reflect this distinction.

[6] aimed at elimination of transitivity axioms from an ontology in order to reduce its expressivity. [1] presented an inference service for approximate translation of a concept from one Description Logic to (typically) less expressive Description Logic. In comparison with our approach, both these approaches center on logical features, while we follow a more engineering-oriented approach, taking into account the view of the human modeller. There is also the approach published in [10], which aims at tractable TBox reasoning over a very expressive Description Logic. They proposed approximate TBox reasoning using EL

[21] On the other hand, we did not check how many of them have forbidden constructs.

rules and additional deduction rules. Transformation of badly tractable constructs are realized as additional data structures. In comparison, our approach addresses general transformation and is centered around the idea of transformation patterns as reusable transformation rules, while reasoning as such is left to reasoning tools. Thus, while in our approach a tool obtains a transformed ontology, in the case of the approach in [10] the transformation is used for an approximate reasoning algorithm and there is no transformed version of an ontology on the output. Furthermore, [10] does not consider nominals replacement and does not remove every non-EL axiom.

Regarding the tricky expressions for a particular reasoner, in [4] there has been presented the lint tool Pellint applicable on ontologies incurring reasoning performance problems to the Pellet reasoner.[22] Particularly, Pellint detects problematic modeling constructs as patterns. There are two groups of patterns: axiom-based patterns dealing with a single axiom, and ontology-based patterns dealing with two or more axioms in the whole ontology. These patterns could be captured by means of our transformation patterns to some extent.

The most prominent project in ontology transformation in general (i.e. aside the simplification setting) is probably OPPL [2], which we introduced in Sec. 2; we directly reuse it in our framework. In [9] the authors consider ontology translation from the Model Driven Engineering perspective. The basic shape of our transformation pattern (as described in detail in [12]) is very similar to their meta-model. However, the transformation is considered at the data level rather than at the schema level as (primarily) in our approach. In [5] the authors presented an ontology update framework that can automatically apply change patterns capturing the evolution of a domain of interest. Their approach is however based on the RDF model and SPARQL update language, while our approach is built on the top of the OWL model.

7 Conclusions and Future Work

This paper presents an approach to ontology construct replacing and complexity downgrading where pattern-based transformation is applied on the source ontology to derive a target ontology. We explained these two use cases and demonstrated their usefulness on examples. We also performed a tiny experiment from the reasoning perspective; the positive effect of our approach was only weak there, which is attributed to the fact that the current replacement transformation patterns only cover a small subset of problematic issues and the ontologies do not contain additional axioms needed for derivation of subsumption relations with replaceable forbidden constructs. The strong point of the presented approach is however that, in contrast to research focused on solving widely the 'notorious' problems of logical inference, the users can easily design their own transformation patterns[23] to address a certain, specific and unforeseen, construct-replacing

[22] `http://clarkparsia.com/pellet/`

[23] Recently a graphical editor of TP authoring has been released as plug-in for Eclipse: `http://owl.vse.cz:8080/tpe/`

use case, such as that specifically dealing with nominals (Sec. 4) or complexity downgrading for a certain, newly introduced profile (Sec. 5). If such patterns are shared, other users could easily apply them through the online transformation web services (i.e. without the necessity to install a particular reasoner as in the logic-centric approaches to transformation).

We plan to investigate what other kinds of transformation patterns and use cases and, moreover, other complexity downgrading tasks, could be addressed by the presented framework. Our approach could be further improved e.g. by precomputing the subsumptions of named classes in the source ontology and adding them into the target ontology. Regarding practical implementation, we plan to extend the support of the framework for analogous datatype-related constructs in OWL such as *DataOneOf*.

This research has been partially supported by the DAAD grant "Pattern-based ontology transformation supporting ontology matching and reasoning tasks" and by CSF grant no. P202/10/1825, "PatOMat – Automation of Ontology Pattern Detection and Exploitation".

References

1. Brandt S., Kuesters R., Turhan A.-Y.: Approximation and Difference in Description Logics. In: 8th Conf. Principles of Knowledge Representation and Reasoning (KR2002), Toulouse.
2. Egaña M., Stevens R., Antezana E.: Transforming the Axiomisation of Ontologies: The Ontology Pre-Processor Language. In: W'shop OWL Experiences and Directions (OWLED 2008 DC), Washington DC.
3. Iannone L., Palmisano I., Rector A., Stevens R.: Assessing the Safety of Knowledge Patterns in OWL Ontologies. In: 7th Extended Semantic Web Conference (ESWC 2010), Heraklion.
4. Lin H., Sirin E.: Pellint - A Performance Lint Tool for Pellet. In: W'shop OWL Experiences and Directions (OWLED 2008), Karlsruhe.
5. Lösch U., Sebastian S., Vrandečić D., Studer R.: Tempus Fugit - Towards an Ontology Update Language. In: 6th European Semantic Web Conference (ESWC 2009), Heraklion.
6. Motik B.: Reasoning in Description Logics using Resolution and Deductive Databases. PhD thesis, Univ. Karlsruhe, 2006.
7. Motik B., Grau B. C., Horrocks I., Wu Z., Fokoue A., Lutz C.: OWL 2 Web Ontology Language Profiles. W3C Recommendation, 2009, online `http://www.w3.org/TR/owl2-profiles/`.
8. Noessner J., Niepert M.: ELOG: A Probabilistic Reasoner for OWL EL. In: 5th Conf. Web Reasoning and Rule Systems (RR 2011), Galway.
9. Parreiras F., Staab S., Schenk S., Winter A.: Model Driven Specification of Ontology Translations. In: 27th Int'l Conf. Conceptual Modelling (ER 2008).
10. Ren Y., Pan J. Z., Zhao Y.: Soundness Preserving Approximation for TBox Reasoning. In: AAAI2010.
11. Svátek V., Šváb-Zamazal O., Vacura M.: Adapting Ontologies to Content Patterns using Transformation Patterns. In: WOP 2010.
12. Šváb-Zamazal O., Svátek V., Iannone L.: Pattern-Based Ontology Transformation Service Exploiting OPPL and OWL-API. In: EKAW 2010.