User-Friendly Pattern-Based Transformation of OWL Ontologies

Ondřej Šváb-Zamazal, Marek Dudáš, and Vojtěch Svátek

Department of Information and Knowledge Engineering,
University of Economics, W. Churchill Sq.4, 130 67 Prague 3, Czech Republic
{ondrej.zamazal|xdudm12|svatek}@vse.cz

1 Motivation for Pattern-Based Ontology Transformation

The high expressivity of the OWL ontology language often allows to express the same conceptualisation in different ways. A simple example is the difference between 'class-centric' and 'property-centric' modelling styles, such that the same notion is modelled as a class in the former (e.g. 'Purchase') and an object property in the latter (e.g. 'bought_from'). Similarly, concept subordination can be expressed via a subclass hierarchy or via individuals connected by a dedicated property (as in SKOS). Such heterogeneity is an obstacle to reusing ontologies in advanced semantic web scenarios. In particular (as mentioned in [1]), two ontologies modelled in different styles are difficult to *match* or to *import* into one another. Furthermore, opting for a style when designing an ontology may have an impact on the applicability and performance of *reasoners*, as some features cause performance problems for certain reasoners. Finally, *human users* may also prefer viewing ontologies in a certain form, possibly 'folding' parts of their complexity. Semi-automatic transformation of the modelling style of existing ontologies, with the help of tools to be presented in the demo, will alleviate such problems.

In the paper we first overview the core framework and then focus on user-oriented tools that allow to perform ontology transformation and edit transformation patterns in a friendly way.

2 *PatOMat* Transformation Framework

The central notion in the *PatOMat* framework[1] 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, except that *placeholders* are allowed in addition to concrete OWL entities. An OP consists of *entity declarations* (of placeholders and/or concrete entities), *axioms* and *naming detection patterns*; the last capture the naming aspect of the

[1] [4] provides more details about the (earlier version of the) framework, and at `http://owl.vse.cz:8080/tutorial/` there is a fully-fledged tutorial for the current version.

OP, which is 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 names for target 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 is available either as a *Java library* or as three *RESTful services*.[2] The Java library is directly used in the GUIPOT and XD Transformation Wizard tools, see Section 3. The whole transformation is divided into three steps that correspond to the three core services:

- *OntologyPatternDetection* service takes the TP and an ontology on input, and returns the binding of entity placeholders on output, in XML. The naming detection patterns of the source OP are first processed. As a result of applying the naming aspect, bound placeholders arise that are placed to the FILTER component of a SPARQL query (generated according to axioms in the source OP) before its execution.
- *InstructionGenerator* service takes the binding of placeholders and the TP on input, and returns transformation instructions on output.
- *OntologyTransformation* service takes transformation instructions and the original ontology on input, and returns the transformed ontology on output.

The third service is partly based on OPPL[3] and partly on our specific implementation over OWL-API.[4] In contrast to plain OPPL, we use naming constraints and we decompose the process of transformation into parts, which enables user intervention within the whole workflow.

3 User-Oriented Tools for Ontology Transformation

The *GUIPOT* (Graphical User Interface for Pattern-based Ontology Transformation) Protégé plugin, see Figure 1, was developed in order to bring ontology transformation into the standard working environment of a knowledge engineer. The screenshot demonstrates how an ontology fragment expressing the notion of 'paper whose decision is rejection' is transformed into an explicit class RejectedPaper, using a generic transformation pattern (with heuristics for naming detection/transformation included). After loading the transformation pattern, GUIPOT displays a list of pattern instances of the source OP detected[5] in the given ontology. Detected pattern instances can be manually adjusted or even new pattern instances can be created before the transformation. By selecting an

[2] All accessible via the web interface at `http://owl.vse.cz:8080/`.
[3] `http://oppl2.sourceforge.net/`
[4] `http://owlapi.sourceforge.net/`
[5] The user can turn on *recursive detection* of pattern occurrences (over a taxonomy), as supported by a reasoner.

instance, the detected entities (placeholder bindings) are highlighted in a classical hierarchy view and also visualized using the OntoGraf plugin[6] on the left part of the plugin window. The right part of the window shows the ontology after transformation, with affected entities indicated by red arrows.

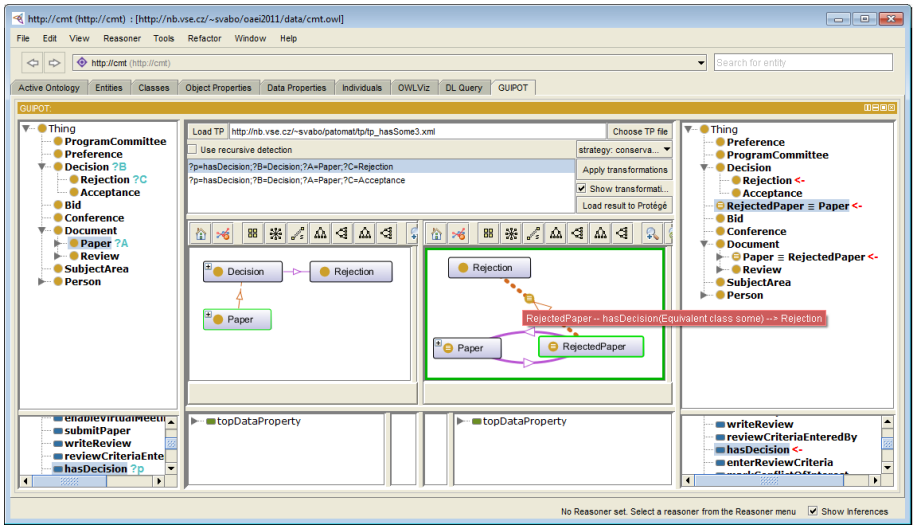

Fig. 1. GUIPOT in action

Aside from GUIPOT as generic graphical interface, we also aimed at support for specific ontological engineering scenarios. One of them is the import of *ontology content patterns* [2] (as small 'best-practice' chunks of ontological knowledge) into legacy ontologies, which, in turn, often have to undergo transformation. For example, when an ontology is to be adapted to the *AgentRole* pattern,[7] often seemingly 'natural' classes have to be changed to 'role' classes. To make such operations rapid and smooth, we decided to closely integrate *PatOMat* with *XDtools*,[8] an ontological engineering environment specifically tailored for ontology content patterns (CPs) exploitation. The tool provides the NeOn toolkit perspective that includes, among others, our *Transformation Wizard*.

When the wizard is invoked, the user chooses the *content pattern* to be imported into an ontology. On the first page of the wizard (see Figure 2) s/he then selects an ontology and a *transformation pattern*. The second page offers the *pattern instances* returned by the detection phase. By clicking on an entity the user can also display its usage within the ontology. The final page of the wizard offers the selection of a finer transformation strategy.

[6] `http://protegewiki.stanford.edu/wiki/OntoGraf`
[7] `http://ontologydesignpatterns.org/wiki/Submissions:AgentRole`
[8] `http://extremedesign.sourceforge.net`

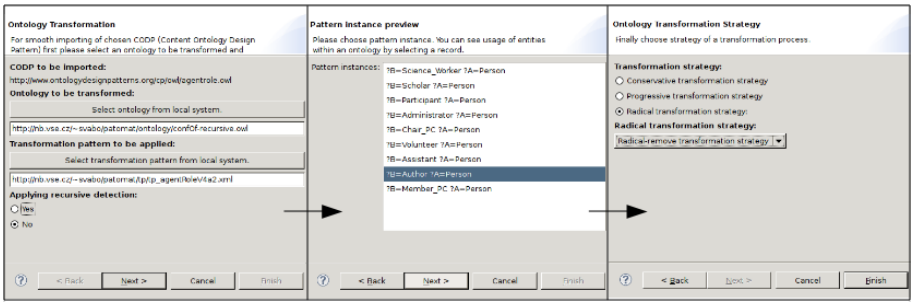

Fig. 2. Step-by-step ontology transformation using XD Transformation Wizard

In order to support the *authoring and update* of transformation patterns, we also developed a *Transformation Pattern Editor* (TPE). It allows for their graphical modeling and export/import from/to the (XML-based) TP notation. TPE is available as a plugin for Eclipse and uses the Graphical Editing Framework.[9] A web-based version is also planned in the near future.

Additional information and installation instructions for GUIPOT, XD Transformation Wizard and TPE are at `http://owl.vse.cz:8080/tools.html`.

4 Demo Summary

The demo[10] will feature several variants of *wizard-based transformation* in NeOn wrt. two best-practice content patterns (for role-based modelling and for reified participation). Other TPs can be applied over *GUIPOT* or via *RESTful services* with simple HTML interface. *TPE* can be shown in the design of a new TP or modification of an existing one. Finally, a dedicated HTML interface to *complexity-downgrading*, i.e. a reasoner-targetted setting of the transformation (not described in the paper due to space limitations), can also be demoed.

The research has been partially supported by the CSF grant no. P202/10/1825. We thank Enrico Daga for his assistance in including PatOMat into XDTools.

References

1. Euzenat J., Shvaiko P.: Ontology matching. Springer, 2007.
2. Presutti V., Gangemi A.: Content ontology design patterns as practical building blocks for web ontologies.: In Proceedings of ER2008. Barcelona, Spain, 2008.
3. Šváb-Zamazal O., Daga E., Dudáš M., Svátek V.: Tools for Pattern-Based Transformation of OWL Ontologies. Presented as demo at ISWC'11, Bonn, 2011.
4. Šváb-Zamazal O., Svátek V., Iannone L.: Pattern-Based Ontology Transformation Service Exploiting OPPL and OWL-API. In: EKAW-2010, Lisbon, Portugal, 2010.

[9] `http://www.eclipse.org/gef/`

[10] This demo paper is successor of [3]. The main enhancements since then are the GUIPOT tool and the NeOn version of the wizard.