PatOMat2: A Tool for Pattern-Based Ontology Transformation using SPARQL

Ondřej Zamazal[1,*], Martin Ledvinka[1,2] and Vojtěch Svátek[1]

[1]Prague University of Economics and Business, Czechia
[2]Faculty of Electrical Engineering, Czech Technical University in Prague

Abstract

The PatOMat2 tool aims at pattern-based transformation of existing ontologies. It allows for bridging different modeling styles of web ontologies. The PatOMat2 tool is a web-based application with a separate backend and frontend. Besides a short introduction to ontology transformation, the demo paper presents the basics of transformation patterns, one of the envisioned use cases, the tool workflow and its architecture.

Keywords

Ontology, Ontology Transformation, Ontology Matching, Ontology Engineering, Semantic Web

1. Introduction and Related Work

Domain knowledge sharing is often made via a domain ontology [1] modeled in the Web Ontology Language (OWL) [2]. Ontology designers usually choose one of many possible ontology representations of the reality [3], reflecting different *modeling styles*. For example, a complex relationship can be expressed as a chain of simpler relationships only (e.g., for a country *hasCapital*, *hasMayor*) or we can also make a shortcut using complex relationship (e.g., to represent the mayor of a country's capital, *hasMayorOfCapital*). Similarly, the fact that a country is a monarchy can be expressed by making it an instance of class *Monarchy* or assigning it a value *monarchy* for the property *hasPolitical System*, or both. The choice of modeling style then impacts stream-line applications that consume the ontology and apply its ontological viewpoint. For example, an instance data *graph visualization* tool may use the chain property (if available) to shrink the displayed graph of country properties, and a *class hierarchy viewer* may make explicit the categorization of countries by political system. Provided the original ontology does not reflect the alternative style patterns yet, we may wish to transform it – usually, but not necessarily, monotonously – to the style desired by the application. Such an *ontology transformation* can possibly be fully automated, or semi-automated, such that the ontology engineer can interact with the process.

One of the first approaches to ontology transformation was OPPL [4, 5]. OPPL, the Ontology Pre-Processing Language, is a macro-language based on Manchester OWL syntax for manipulating ontologies written in OWL. OPPL is based on OWL API and does not allow human interaction. The PatOMat project [6] proposed automated transformation with a three-step workflow: pattern occurrence detection, transformation instructions generation, and actual transformation. The implemented tool was based on OWL API and equipped with a graphical user interface as a plugin for Protégé. Several pilot use cases were demonstrated [7]. Another approach, EvoPat [8], was based on the SPARQL language and targeted not only schema but also instance data transformation. It did not address high-level logical patterns and entity naming.

The PatOMat2 prototype tool, presented in this paper, is being designed so as to enable both a fully automated and semi-automated mode of ontology transformation. PatOMat2 features multiple

EKAW 2024: EKAW 2024 Workshops, Tutorials, Posters and Demos, 24th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2024), November 26-28, 2024, Amsterdam, The Netherlands.
*Corresponding author.
ondrej.zamazal@vse.cz (O. Zamazal); martin.ledvinka@fel.cvut.cz (M. Ledvinka); svatek@vse.cz (V. Svátek)
https://nb.vse.cz/~svabo/ (O. Zamazal); https://nb.vse.cz/~svatek/ (V. Svátek)
0000-0002-7442-9016 (O. Zamazal); 0000-0002-2256-2982 (V. Svátek)

enhancements compared to the original PatOMat [9], on which it partly follows up:

- The original PatOMat supported OWL 2 constructs using the Manchester syntax, which allowed to easily express complex concepts. Conversely, more complex language also means a more complicated way of parsing and transformation instructions generation. For PatOMat2, we expect less complex concepts, but a higher need for many relationships where RDF triples are sufficient. Thus, PatOMat2 is based on a straightforward approach where SPARQL query and SPARQL Update are used.
- PatOMat2 comes as a web application, so it is easy to use and does not require the user to download or install anything. In contrast, the original PatOMat was developed as a Protégé plugin. The necessity of installing all dependencies related to, e.g., NLP modules most likely hindered its wider adoption.
- Not only does PatOMat2 allow users to select the pattern instances to apply in the transformation (as did the original PatOMat), but it also allows the users to edit the labels generated for new entities.
- Since it does not depend on another tool, PatOMat2 is easier to evolve and extend.

In the rest of the paper we first present a concrete motivating use case, and then explain the functionality of the tool on an example reflecting the use case; finally, implementation details are given.

2. Motivation Use Case: Rule Mining in the Circular Economy Domain

The ontology engineering methods researched and deployed in the EU Onto-DESIDE[1] project aim to support the distributed management of information about the flow of entities (materials, products, etc.) within the *circular economy* (CE) pathways [10], as well as analytical tasks on top of this flow.

For example, the Building Product Ontology (BPO)[2] focuses on how different product components can be assembled. Let us consider a *rule mining* task over (relational) data corresponding to BPO and other interlinked ontologies. A possible output rule could be one capturing the situation that if an assembly contains a *dynamic* entity (i.e., by BPO, an entity representing multiple components that are multiple, undistinguished, and sharing all their features – e.g., identical legs of furniture), it could be better reused in terms of such its parts. The corresponding rule could look like as follows: *Assembly(?x) & isComposedOfEntity(?x,?y) & DynamicEntity(?y)* $\implies$ *PartwiseReused(?x)*. However, the rule mining templates are often limited to a certain number of atoms in order to constrain the search space, and such a long rule might not be found. On the other hand, if we transform the antecedent part of the rule to a single atom, it becomes much more compact and thus easier to discover: *AssemblyWithDynamicEntity(?x)* $\implies$ *PartwiseReused(?x)*. In data mining (particularly, inductive logic programming as its subfield), methods for such *feature construction* already exist [11]. However, they are applied at the level of the data mining task only, thus are ignorant of the ontology structure as a whole, do not feed back to it, and the new entities produced ('invented predicates') are usually not endowed with any meaningful naming. On the other hand, if we make the transformation (add the new concept) at the ontology level, using a transformation pattern, we get a reusable and documented solution for the same task. This is what we demonstrate in the workflow description in the following section.

3. PatOMat2 Workflow and Architecture

In PatOMat2, ontology transformation is based on *transformation patterns*. Each transformation pattern consists of three parts: a source ontology pattern, a target ontology pattern and a naming transformation. An *ontology pattern* (OP) is an OWL ontology fragment with variables, expressible as a set of triple

[1] https://ontodeside.eu/
[2] https://annawagner.github.io/bpo/

patterns forming a SPARQL query pattern. A *naming transformation* represents the semantic link between the source and target pattern in lexical terms. Its nature can range from simple regex operations through complex symbolic templates referencing lexical databases (e.g., WordNet) interfaces to pre-trained or prompted large language models (LLMs), although the advanced methods are not yet fully implemented in the tool. For the symbolic template approach, the instructions contain a combination of text, variables from triple patterns, and naming instructions applied to variables. Currently supported simple naming operations are:

- *label(?X)*: get the label of the entity if available, otherwise return the local IRI fragment
- *nominalize(?X)*: converting the verb or adjective of the entity name to a noun form
- *passivize(?X)*: converting the tense of the verb of the entity name into the passive one
- *head_noun(?X)*: get the head (main) part of the entity name

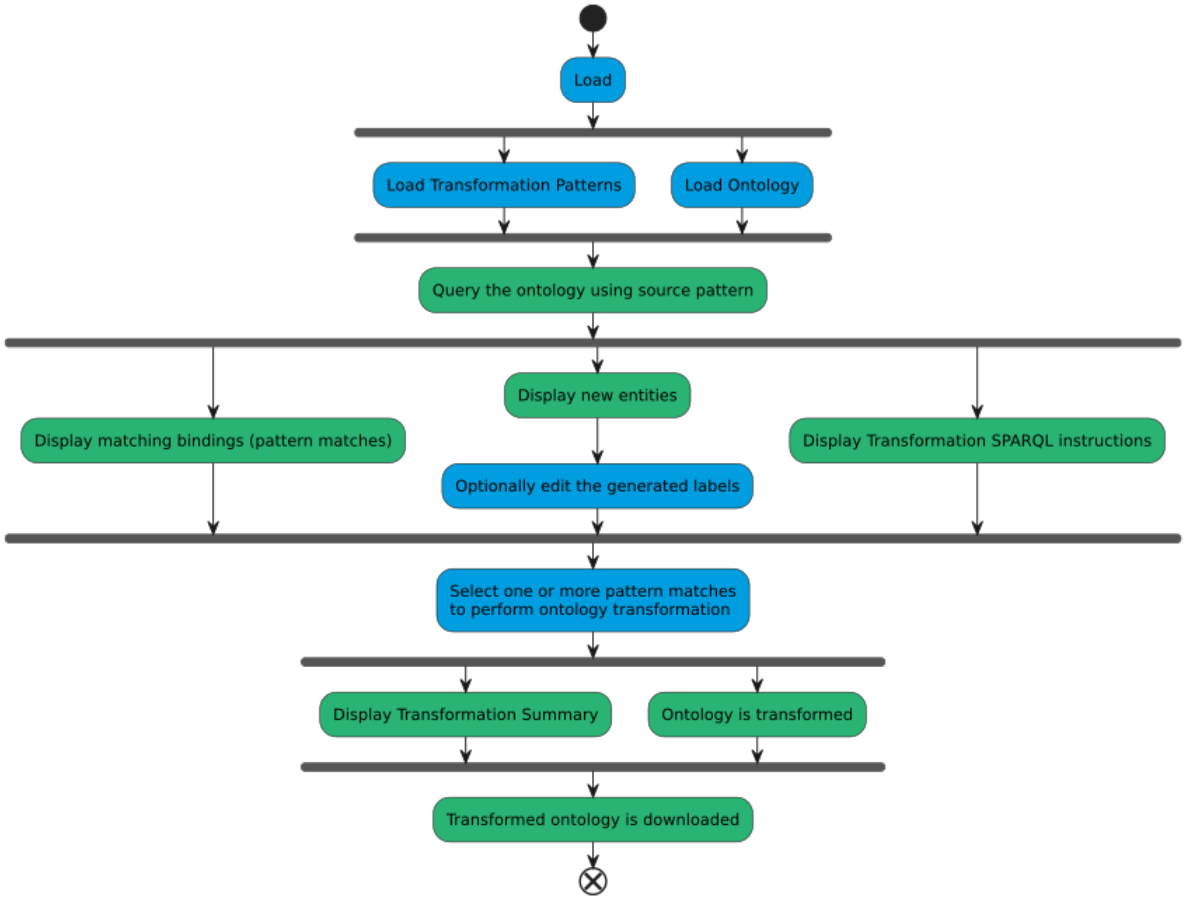

Figure 1: PatOMat2 Workflow. Blue parts are user activities, while green parts are automatic activities.

Workflow. The overall workflow is depicted in Figure 1. The ontology and the transformation patterns (represented in JSON) first need to be loaded. For example, the CAT ('class by attribute type') transformation pattern[3] materializes, for a certain object property p, a subclass G of its domain class A depending on an existing subclass C of its range class B. In JSON, the source and target ontology patterns of CAT look as follows:

```
"op_source": {
    "triples": {
        "triple": [
```

[3]https://github.com/Onto-DESIDE-VSE/TransformationPatterns/blob/main/submissions/CAT1ntp-tp-lite2x.json

```
            "?p rdfs:domain ?A",
            "?p rdfs:range ?B",
            "?C rdfs:subClassOf* ?B"
        ]
    },
    "filters": { "filter": [ "FILTER(?A != ?B)" ] }
}

"op_target": {
    "triples": {
        "triple": [
            "?p rdfs:domain ?A",
            "?p rdfs:range ?B",
            "?C rdfs:subClassOf ?B",
            "?G rdfs:subClassOf ?A",
            "?G owl:equivalentClass _:restriction",
            "_:restriction rdf:type owl:Restriction",
            "_:restriction owl:onProperty ?p",
            "_:restriction owl:someValuesFrom ?C"
        ]
    }
}
```

In the current implementation, CAT also contains a baseline naming transformation property, with instructions on how to generate the required new label. For the new entity `?G` it looks as follows:

```
"ntp": {
    "?G": [
        "label(?A) that label(?p) a label(?C)"
    ]
}
```

Initially, the application queries the ontology using SPARQL based on the source pattern. The pattern definition can use various SPARQL features such as property paths (here, `rdfs:subClassOf*` for transitive subsumption) and filters (here, `?A!=?B`). The detection results are displayed as *matching bindings* and as *new entities*. For example, when applying CAT on the BPO ontology mentioned in Section 2, we obtain one matching binding for each variable, `?p`, `?B`, `?A` and `?C` from the source pattern:

- `?p` = `<https://w3id.org/bpo#isComposedEntity>`
- `?B` = `<https://w3id.org/bpo#Entity>`
- `?A` = `<https://w3id.org/bpo#Assembly>`
- `?C` = `<https://w3id.org/bpo#DynamicEntity>`

New entities are displayed with their randomly generated IRIs and labels generated based on NTP (Naming Transformation Pattern) instructions. In our use case (application of CAT on BPO), there will be a new (class) entity for `?G` with its provisional label currently generated symbolically as *Assembly that is composed of entity a dynamic entity*. (However, as we have checked, [12] mere pre-trained off-the-shelf LLMs can easily come up with more compact labels such as *Assembly with dynamic entity* from the motivating example.) Next, the user can directly edit the generated label.

The user can select one or more of the displayed *pattern matches* (and apply filtering on the uploaded TPs) to perform *ontology transformation*. Based on the transformation pattern, the transformation can be done either by *inserting triples* (option “Apply only inserts”) or by *inserting and deleting triples* (option “Apply deletes and inserts”). The information about the transformation is presented under

Transformation SPARQL, where the SPARQL UPDATE statements can be found. Finally, a summary of the changes applied is displayed when the transformation has been performed and the transformed ontology has been downloaded.

System Architecture. PatOMat2 is a web application with a separate backend and frontend. The backend is written in Java using Spring Boot. Its internal architecture follows the *domain-driven approach* [13], where domain entities contain both data and the relevant processing logic. The backend exposes a REST API that the frontend (and possibly other clients) can use. Documentation for the REST API is generated using the OpenAPI standard[4] and is accessible directly from the instance. The backend uses RDF4J [14] to work with the ontological data. However, its use is separated from the application logic by generic interfaces that allow, if need be (for example, due to higher expressiveness of the data), switching to a different ontology processing library (like Jena [15] or OWL API [16]). The frontend is written in TypeScript using Vue with Material UI for some basic styling.

4. Conclusions and Future Work

The demo paper presents the PatOMat2 tool aiming at pattern-based ontology transformation. We presented it by explaining the transformation pattern, workflow, system architecture, and using a simple running example related to data mining on circular economy data. While the current version of the tool is more suitable for knowledge engineers, we strive to support subject-matter experts by enhancing the graphical user interface in the future. Since people will use transformed ontologies, meaningful labels are essential. We plan to include LLMs for label or even comment generation. In the future, we also envisage supporting instance data translation based on the transformed ontology. Finally, we foresee functional testing of the tool and overall user experience evaluation.

Acknowledgments

This work has been supported by the EU's Horizon Europe grant no. 101058682 (Onto-DESIDE).

References

[1] T. R. Gruber, Toward principles for the design of ontologies used for knowledge sharing?, International journal of human-computer studies 43 (1995) 907–928.

[2] W. W. W. Consortium, et al., Owl 2 web ontology language document overview, Word Wide Web Consortium (2012).

[3] C. Shimizu, K. Hammar, P. Hitzler, Modular ontology modeling, Semantic Web 14 (2023) 459–489. URL: https://doi.org/10.3233/SW-222886. doi:10.3233/SW-222886.

[4] M. Egana, A. Rector, R. Stevens, E. Antezana, Applying ontology design patterns in bio-ontologies, in: Knowledge Engineering: Practice and Patterns: 16th International Conference, EKAW 2008, Acitrezza, Italy, September 29-October 2, 2008. Proceedings 16, Springer, 2008, pp. 7–16.

[5] L. Iannone, M. E. Aranguren, A. L. Rector, R. Stevens, Augmenting the expressivity of the ontology pre-processor language., in: OWLED, volume 432, 2008.

[6] O. Šváb-Zamazal, V. Svátek, L. Iannone, Pattern-based ontology transformation service exploiting oppl and owl-api, in: Knowledge Engineering and Management by the Masses: 17th International Conference, EKAW 2010, Lisbon, Portugal, October 11-15, 2010. Proceedings 17, Springer, 2010, pp. 105–119.

[7] O. Zamazal, V. Svátek, Patomat-versatile framework for pattern-based ontology transformation, Computing and Informatics 34 (2015) 305–336.

[8] C. Rieß, N. Heino, S. Tramp, S. Auer, Evopat–pattern-based evolution and refactoring of rdf knowledge bases, in: The Semantic Web–ISWC 2010: 9th International Semantic Web Conference,

[4] https://spec.openapis.org/oas/latest.html, accessed November 19, 2024

ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part I 9, Springer, 2010, pp. 647–662.
[9] O. Zamazal, V. Svátek, Patomat - versatile framework for pattern-based ontology transformation, Comput. Informatics 34 (2015) 305–336. URL: http://www.cai.sk/ojs/index.php/cai/article/view/1138.
[10] E. Parliament, Circular economy: definition, importance and benefits, 2017.
[11] O. Kuzelka, F. Zelezný, Block-wise construction of tree-like relational features with monotone reducibility and redundancy, Mach. Learn. 83 (2011) 163–192. URL: https://doi.org/10.1007/s10994-010-5208-5. doi:10.1007/S10994-010-5208-5.
[12] V. Svátek, O. Zamazal, K. Haniková, D. Chudán, M. J. Saeedizade, E. Blomqvist, Welcome, newborn entity! On handling newly generated entities in ontology transformation, in: I. Novalija, C. Badenes-Olmedo (Eds.), Companion Proceedings of the 24th International Conference on Knowledge Engineering and Knowledge Management, Amsterdam, Netherlands, CEUR Workshop Proceedings, To Appear. CEUR-WS.org, 2024.
[13] E. Evans, Domain-driven design: tackling complexity in the heart of software, Addison-Wesley Professional, 2004.
[14] J. Broekstra, A. Kampman, F. Van Harmelen, Sesame: A generic architecture for storing and querying rdf and rdf schema, in: International semantic web conference, Springer, 2002, pp. 54–68.
[15] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, K. Wilkinson, Jena: Implementing the Semantic Web Recommendations, in: Proceedings of the 13th international World Wide Web conference (Alternate Track Papers & Posters), 2004, pp. 74–83.
[16] M. Horridge, S. Bechhofer, The OWL API: A Java API for OWL ontologies, Semantic Web – Interoperability, Usability, Applicability (2011).

A. Online Resources

The following resources are available:

- The instance of the tool, https://owl.vse.cz/patomat2
- The screencast for the EKAW 2024 demo, https://owl.vse.cz/patomat2/about
- The code, https://github.com/Onto-DESIDE-VSE/patomat2
- The Documentation for the REST API, https://owl.vse.cz/patomat2/service/server/swagger-ui/index.html