# ODPReco - A Tool to Recommend Ontology Design Patterns *

Maleeha Arif Yasvi[0000−0001−8155−8804] and Raghava Mutharaju[0000−0003−2421−3935]

Knowledgeable Computing and Reasoning Lab,
IIIT-Delhi, India
{maleeha18112,raghava.mutharaju}@iiitd.ac.in

**Abstract.** Ontologies evolve over time due to changes in the domain and requirements of the application. Maintaining an ontology over time and keeping it up-to-date with respect to the changes in the domain and requirements of application is hard. But a high quality ontology can significantly reduce the effort and cost of ontology maintenance. Ontology Design Patterns (ODPs) can be used to improve the quality of an ontology and make it more modular and reusable. But with around 220 (and increasing) ODPs across six different categories, it is not easy to determine the right set of ODPs to choose for a particular use case even for experts. This becomes even more difficult in the case of refactoring existing ontologies using the right set of ODPs. We describe here a proposal for a work-in-progress tool named *ODPReco* that can recommend the possible ODPs to use in a given ontology. ODPReco analyzes the lexical, structural, and behavioural aspects of an ontology, along with learning from existing ODP implementations to recommend ODPs that can be used for refactoring an ontology.

**Keywords:** Ontology Design Pattern · Ontology Maintenance · Refactoring Ontologies · Modular Ontologies

## 1 Introduction

Ontologies capture the state of the world (domain) at the point of their creation. But over time, there will be changes in the domain and the applications that depend on the ontology will evolve as well. So an ontology is not a static piece of artifact that can be built once and used over the lifetime of the application. It needs to evolve with the changes in the domain and application requirements. The developers maintaining the ontology may change over time and they might want to make changes to the class hierarchy, relationships among the classes, make some classes as properties etc. There is a body of work on ontology evolution [14, 2] and maintenance. One of the crucial factors that is always stressed upon is that, the quality of the ontology plays a very important role in ontology

maintenance. A good quality ontology is modular, easy to understand, and more amenable to changes [7]. This makes it easier to maintain the ontology, which in turn reduces the maintenance cost. This holds true for the maintenance of software artifacts as well. It is said that around 80% of the software development cost goes into maintenance and a good quality software can significantly reduce the cost [1].

Ontology Design Patterns (ODPs) [6] are solutions to common modelling problems. They are generally small in size and can be used as building blocks in developing modular ontologies. Large monolithic ontologies can be refactored to use ODPs and thus improve their quality. But it is not easy for ontology developers, especially the inexperienced ones to identify the right ODPs to use for a particular ontology (or use case that the ontology is modelled for). This is due to the number and variety of ODPs that are available. ODP repository[1] lists nearly 220 (and counting) ODPs spread across six different categories. We propose a work-in-progress tool named *ODPReco* that analyzes the lexical, structural, and behavioural aspects of an ontology to recommend a set of ODPs that can be used to refactor a non-modular ontology. ODPReco also learns from existing ODP implementations. If a non-modular ontology (or part of it) is *similar* to an existing ontology that has implemented ODPs, then the non-modular ontology can also make use of the same ODPs.

## 2 Approach

In order to recommend ODPs, ODPReco would need a list of all the available ODPs including their details such as the description, competency questions, and use cases. Along with that, a collection of ontologies in which the ODPs used are clearly indicated is also required. This helps the tool to learn from existing ODP implementations. The following two datasets satisfy these requirements. We will refer to these datasets as our *collection*.

a) ODPs from the ODP repository. We will have a local copy of all the ODPs from the repository and create an index based on the details of each ODP such as the name, description, competency questions, use cases, similar patterns, class names, and property names.
b) MODL: Modular Ontology Design Library [12]. It is a downloadable curated collection of well-documented ODPs. It contains annotations indicating which patterns were used and also the axioms that are part of the pattern.

We will discuss two complementary techniques to ODP recommendation in the following sections.

### 2.1 Ontology Analysis

We will be analyzing a given ontology on the following dimensions.

[1] http://ontologydesignpatterns.org/

a) **Lexical Analysis**. The names of classes, properties, and instances from the ontology are compared with the ones from our collection. Apart from names, we will be including other descriptive text such as labels and comments. Word embeddings can be used to get a ranked list of similar words (and hence ODPs that can contain these words) present in our collection.
b) **Structural Analysis**. We will be comparing the axioms of the given ontology with the ones from our collection. We can generate embeddings for the axioms by converting them into a fixed format. An example for axioms involving cardinalities is given below. Each axiom type will have a format along the same lines.

   ⟨AxiomTypeID⟩ ⟨ClassesInAxiom⟩ ⟨PropertiesInAxiom⟩ ⟨Cardinality⟩

   Using the embeddings generated from the given ontology as well as the ones from our collection, we will generate similarity scores and rank the axioms (and the corresponding ODPs that are associated with these axioms).
c) **Behavioural Analysis**. The behavioural aspect of the ontology can be analyzed by making use of the competency questions associated with the given ontology. They represent the ontology requirements and help in capturing the scope an ontology. These competency questions are compared with the ones associated with the ODPs and a similarity score is generated. We are aware of the fact that most ontologies do not have competency questions associated with them. But if available, we will analyze the behaviour of the ontology using the competency questions.

The scores obtained from these three dimensions of an ontology are integrated and normalized. Weights can also be assigned to each of these three dimensions. Depending on the scores, the top $k$ ODPs can be recommended to the user for the given ontology.

### 2.2 Supervised Machine Learning on Existing ODPs

Chess [8] and cooking recipes [11] ontologies along with the MODL collection can be considered as labeled data. The ODPs used in these ontologies are clearly known and they are well documented. This can be used as training data. During the training phase, we should capture features that can help in determining why a particular ODP has been used in an ontology. The three types of analysis discussed in the previous section - lexical, structural, and behavioural aspects of the ontology can be used as features. There could be other aspects of an ontology that can be used as features, but we have not yet looked into all the possibilities. After learning the machine learning model, we can now use the given ontology to predict the classes (ODPs) it should belong to. So this becomes a multi-label classification problem. But there are some obvious disadvantages to using machine learning in this context.

- Training data is limited.

- The coverage of the training data is limited, i.e., not all possible 220 ODPs are covered in the training data. So there will be a bias towards the ODPs that are most frequently used in the training data.
- There are large number of classes (220). So prediction will not be accurate.

Due to these reasons, it is important to complement the machine learning model with an alternative, which is to predict (recommend) ODPs by analyzing the ontology.

We are currently implementing the ideas discussed here. ODPReco will be available publicly and it could also be made into a protégé plug-in.

## 3 Related Work

In the software engineering community, there has been work on recommending software design patterns by analysing the code. We plan to adapt that work to the case of recommending ODPs for a given ontology. We briefly discuss some of the techniques used for recommending software design patterns. Along with this, we will also discuss work related to ODPs.

In [13], code smells (characteristics of the code) are analyzed to recommend software design patterns. Design pattern recommendation systems are developed using various techniques. One such technique is the text based approach [5]. In this approach, the description and the scenario of the design patterns provided are preprocessed. Pre-processing includes tokenization, stop word removal and stemming. Vector space model is used to represent the collection of design patterns in the form of unigrams and bigrams and the most suitable design pattern for a given problem scenario is selected which is based on the cosine similarity and TF-IDF. Another approach for recommending patterns is based on question-answering [10]. It is an interactive approach in which questions are asked from the user and the user answers them with a yes/no/do not know. Based on these answers, weights are assigned and software design is predicted. In [9], case based reasoning is used. Structural, behavioural and semantic aspects are considered in [3] to recommend patterns. The structural aspects are generalization, attribute, aggregation, and specialization methods. The behaviour of the design pattern can be analyzed through the control flow graphs [13]. The semantic aspect can be analyzed using the programming guidelines and naming conventions. Software design patterns are also recommended using the classification techniques [4].

Ontology Design Patterns [6] are small, self-contained ontologies that provide a solution to a commonly occurring modelling problem across different domains. ODP repository has around 220 ODPs that are divided into six categories, structural, reasoning, correspondence, presentation, lexico-syntactic and content ODPs. MODL [12] is a well documented ODP library that contains ontologies which are clearly annotated with the ODPs.

Even for experienced ontology developers, it is hard to know which among the 220 patterns is a good fit for a given use case. This task becomes hard for inexperienced ontology developers. Our tool, ODPReco, attempts to fill this gap

by learning from existing ODP implementations and analyzing the structure of ontologies to recommend ODPs.

## References


1. AL-Badareen, A.B., et. al.: The Impact of Software Quality on Maintenance Process. International Journal of Computers **5**, 183–190 (2011), http://www.naun.org/main/NAUN/computers/19-862.pdf
2. Djedidi, R., Aufaure, M.A.: Ontology Evolution: State of the Art and Future Directions. Ontology Theory, Management and Design: Advanced Tools and Models pp. 179–207 (2010). https://doi.org/10.4018/978-1-61520-859-3.ch008
3. Dong, J., Zhao, Y., Peng, T.: A Review of Design Pattern Mining Techniques. International Journal of Software Engineering and Knowledge Engineering **19**, 823–855 (2009). https://doi.org/10.1142/S021819400900443X
4. Dwivedi, A.K., Tirkey, A., Rath, S.K.: Software design pattern mining using classification-based techniques. Frontiers of Computer Science **12**(5), 908–922 (Oct 2018). https://doi.org/10.1007/s11704-017-6424-y
5. Hamdy, A., Elsayed, M.: Automatic Recommendation of Software Design Patterns: Text Retrieval Approach. JSW **13**, 260–268 (2018). https://doi.org/10.17706/jsw.13.4.260-268
6. Hitzler, P., et. al.: Ontology Engineering with Ontology Design Patterns: Foundations and Applications. IOS Press, Amsterdam, The Netherlands (2016)
7. Hitzler, P., Shimizu, C.: Modular Ontologies as a Bridge Between Human Conceptualization and Data. In: Graph-Based Representation and Reasoning - 23rd International Conference on Conceptual Structures, ICCS 2018, Edinburgh, UK. Lecture Notes in Computer Science, vol. 10872, pp. 3–6. Springer (2018). https://doi.org/10.1007/978-3-319-91379-7_1
8. Krisnadhi, A., Hitzler, P.: Modeling with ontology design patterns:chess games as a worked example. In: Ontology Engineering with Ontology Design Patterns (2016). https://doi.org/10.3233/978-1-61499-676-7-3
9. Nahar, N., Sakib, K.: Automatic Recommendation of Software Design Patterns Using Anti-patterns in the Design Phase: A Case Study on Abstract Factory p. 8 (2015)
10. Palma, F., Farzin, H., Guhneuc, Y., Moha, N.: Recommendation system for design patterns in software development: An DPR overview. In: 2012 Third International Workshop on Recommendation Systems for Software Engineering (RSSE). pp. 1–5 (Jun 2012). https://doi.org/10.1109/RSSE.2012.6233399
11. Sam, M., Krisnadhi, A.A., Wang, C., Gallagher, J., Hitzler, P.: An Ontology Design Pattern for Cooking Recipes: Classroom Created. In: Proceedings of the 5th International Conference on Ontology and Semantic Web Patterns-Volume 1302. pp. 49–60. CEUR-WS.org (2014), http://dl.acm.org/citation.cfm?id=2878937.2878943
12. Shimizu, C., Hirt, Q., Hitzler, P.: MODL: A Modular Ontology Design Library (2019), https://arxiv.org/pdf/1904.05405.pdf
13. Smith, S., Plante, D.R.: Dynamically recommending design patterns. In: SEKE (2012), https://pdfs.semanticscholar.org/9ea6/731b7a517.pdf
14. Zablith, F., Antoniou, G., d'Aquin, M., Flouris, G., Kondylakis, H., Motta, E., Plexousakis, D., Sabou, M.: Ontology evolution: a process-centric survey. The Knowledge Engineering Review **30**(1), 45–75 (Jan 2015). https://doi.org/10.1017/S0269888913000349