# Ontology Patterns and Beyond
## Towards a Universal Pattern Language

Olaf Noppens and Thorsten Liebig

Inst. of Artificial Intelligence, Ulm University, Ulm, Germany
`firstname.lastname@uni-ulm.de`

**Abstract.** In this paper we argue for a broader view of ontology patterns and therefore present different use-cases where drawbacks of the current declarative pattern languages can be seen. We also discuss use-cases where a declarative pattern approach can replace procedural-coded ontology patterns. With previous work on an ontology pattern language in mind we argue for a general pattern language.

## 1 Motivation

Though ontology engineers are supported by a wide range of authoring tools the task of designing ontologies is still difficult even for experts. Several studies on analyzing common errors in formalizing knowledge may serve as a prime example (e. g. [1], [2]). They have shown that often modeling problems were concerned with the act of writing down conflicting axioms, or with finding solutions for non-obvious but common modeling challenges (e. g. n-ary relationships). Extending the original notion of knowledge patterns [3] ontology design patterns (ODP) provide a modeling solution to solve a recurring ontology design problem [4], [5]. There is a wide range of ODP types such as ontology content patterns (OCPs), logical patterns, architectural patterns etc. [6].

In this paper, we would like to broaden the term *ontology pattern* by looking at different ontology engineering approaches. Some of them rely on procedural knowledge. We argue that lifting them to a declarative level would enable a (semi-)automatic handling of patterns with the aim to support the ontology engineer during the entire ontology's life-cycle. The instantiation of an ontology pattern, e. g., a logical pattern, also depends on conditions that must be fulfilled. Violating these conditions at a later stage typically results in an incorrect instantiation. For that we argue that monitoring pattern instantiations is crucial. In our opinion, there is also a duality between finding pattern instances and instantiating patterns. For instance, finding axioms that correspond to a logical ontology pattern without having explicitly applied that pattern could be useful during (distributed) ontology engineering for better understanding or finding undesired modeling issues. Only little work has been spent in guiding the ontology engineer in finding patterns and monitoring correct instantiation of them.

Recently some work has been done with respect to collaborative and distributed ontology engineering [6] as well as in the context of Protégé 4 [7].

The reminder of this paper is organized as follows: first we discuss use-cases of the application of ontology patterns where some are hidden in procedural knowledge. Then we present a proposal for the obtained requirements and we will briefly discuss the integration in our language in an ontology authoring tool.

## 2 Use-Cases and Requirements

We will try to broaden the term *ontology pattern* by looking at different ontology engineering support methods to extract commonalities that should be fulfilled by a universal ontology pattern language in order to encode patterns in a declarative way. These use-cases are mainly driven by our experience in ontology authoring and therefore not intended to be exhaustive. As a starting point we take into account a recently proposed pattern language [7] based on OPPL. We refer to it as the *Manchester Ontology Pattern Language* (MPL). To the best of our knowledge, it is the only ontology pattern language.

**Ontology Design Patterns** Repositories such as `ontologydesignpatterns.org` provide access to pattern catalogues. Instead of encoding patterns in a declarative way they are described by their intent, examples and diagrams. Recently, the authors of MPL propose to describe patterns in a declarative way to support a more automatic handling of patterns. In the following we observe two shortcomings of their approach. Consider a very simple partition pattern: a *partition* is a structure that is divided into several disjoint parts. Using the Abstract Syntax of OWL 2 [8] it can be specified by the following axioms where $P$ is the partition class, and $C_1$, ..., $Cn$ are arbitrary classes or class expressions:

```
EquivalentClasses(P ObjectUnionOf(C1, C2, ..., Cn))
DisjointClasses (C1, C2, ..., Cn)
```

The first axiom can be expressed in MPL using `createUnion(?x.VALUES)` to bind an arbitrary number of classes to a given variable `?x`. However, MPL does not allow to express a set of classes as needed in the second axiom [1]. Moreover, OPPL/MPL allow variables of named entities only. This limits the usage of the language because some situations (as shown before) cannot be expressed.

To sum up a general pattern language has to deal also with an arbitrary number of classes in conjunction with all axioms in OWL which allow a set of classes (unions, intersections as in MPL but also disjointness, equivalences etc.). All types of OWL objects should be used as variable parts. Otherwise, the very simple partition example would only contain named classes.

[1] We refer here to the MPL grammar available at `http://www.cs.man.ac.uk/?iannonel/oppl/grammar.html`, last accessed on 20th of July, 2009, and [7].

**Refactoring/Re-engineering** There are different syntactical forms to model the same logical meaning in OWL. One reason is that syntactical sugar makes things easier to read and to express. For instance, property range restrictions can be expressed either as `ObjectPropertyRange(hasParent Person)` or using a GCI `SubClassOf(Thing ObjectAllValuesFrom(hasParent Person))`. The latter axiom, for instance, occurs in ontology transformations from KRSS to OWL. Negative property assertions can be either expressed as `NegativeObject-PropertyAssertion(property i1 i2)` or as `SubClassOf(ObjectOneOf(i1), ObjectComplementOf(ObjectSomeValuesFrom(property ObjectOneOf(i2))))` whereas the former one has been recently introduced in OWL 2. Obviously, in both cases, the first statement is more intuitive and convenient and even experts seem to encounter problems in understanding the latter one. It is, of course, an extreme case but it illustrates the problem of understanding the meaning when users look at ontologies created by others. It is not a rocket science to transfer these axioms into each other by writing some code. However, this is not an adequate solution: the transformation is hidden in procedural knowledge. Another simple re-factoring example is the replacement of cyclic subclass axioms with an equivalent-class axiom. `SubClassOf(C0 C1),..., SubClassOf(Cn C0)` is equivalent to `EquivalentClasses(C0,...,Cn)`. This is, for instance, implemented in a procedural manner in Protégé. This example shows that a pattern language needs also to support variability on axiom level: Here, the number of subclass axioms is unknown during pattern design-time.

**Ontology Lint Tools** In Software Engineering lint tools perform static analysis of source code in order to detect suspicious scopes of code wrt. language usage, condition violations, violation of type ranges etc. that might lead to unexpected behavior on run-time. Transferring this idea to ontologies, ontology lint tools give hints about potential modeling defects, i. e., they scan for “anti-patterns” that should be avoided. In our opinion, these anti-patterns do not differ from ontology patterns and therefore they should also be expressed in a declarative way – re-usable independent of any programming language. For instance, Pellint [9] tries to detect modeling patterns which potentially will slow down reasoning performance wrt. a tableaux-based reasoner, mainly optimized for the Pellet reasoner. There is also a tentative ontology lint plugin[2] for Protégé 4 providing several default lints similar to those of Pellint. However, in both tools lints are expressed by procedural knowledge, i. e., the analysis methods are encoded in the program. In contrast to general ontology patterns, additional constraints are typically needed: Pellint defines, for instance, a lint pattern that finds all classes with more than 5 explicit asserted subclass axioms for a given class. This cannot be expressed in MPL because neither a condition nor an arbitrary number of axioms (greater than 5) can be expressed. To sum up, a flexible way to use both an arbitrary number of axioms and conditions is needed. Ontology Lints represents the missing link between finding pattern instances, monitoring as well as instantiating them.

[2] `http://www.cs.man.ac.uk/~iannonel/lintRoll`

## 3 A Proposal towards a Universal Axiom Pattern Language

Inspired by the mentioned use-cases and the shortcomings of MPL we now present building blocks of a pattern language which fulfills the discussed requirements. We do not claim that those requirements are the only ones but they are a good starting point. For the rest of this paper we refer to OWL 2, OWL for short, and its *Abstract Syntax* for presentation purpose. Allowing arbitrary class expressions as possible bindings for variables results in loosing decidability because models of OWL ontologies, in general, are infinite. Therefore, we use the finite set of class (property) expressions as introduced in [10]: given an ontology $\mathcal{O}$, the finite set of class (property) expressions consists of all class (property) expressions that appear in the asserted axioms of $\mathcal{O}$.

**Definition 1 (Variables).** *Class expressions with variables are defined analogously to OWL class expressions but allowing variables at positions of class expressions. The range of a variable can either be a named class, class expression, or any subtype of it as produced by the corresponding rule in OWL. Properties, individuals, and constants are defined analogously. A* variable *is either a* single variable *`?x` that can be bound to a single expression (according to its range), or a* multi-variable *`??X` which consists of a set of single variables where each of them is referable via an index. By default, different index values denote different single variables within the set.*

*Multi-variables* allow to define an arbitrary number of expressions without loosing reference to the single variable if needed. Let `??X` be a multi-variable whose range are classes. Then `ObjectIntersectionOf(??X)` defines an intersection of an arbitrary number of classes. In contrast to the MPL statement `createIntersection(?x.VALUES)`, each `?X(i)` is referable in other condition which have an impact on the variable binding. For instance, one could state that `?X(i)` and `?X(j)` are used in `SubClassOf` axioms: `SubClassOf(?X(i) A)` and `SubClassOf(B ?X(j))` to define further constraints. By default, `?X(i)` and `?X(j)` have different bindings if there is no constraint specifying that both should be equal.

**Definition 2 (Variablized Axioms).** *Given an OWL ontology, axioms are defined analogously to OWL axioms according to the axiom rules, but also allow variables at position of class (resp. property) expressions (resp. indivduals). Axioms with variables are called* variablized axioms. *The semantic of using multi-variables in axioms is the following: for each bounded `?X(i)` a new re-incarnation of the axiom is produced.*

Given an ontology containing the following axioms `SubClassOf(A B)`, `SubClassOf(A C)`, `SubClassOf(A D)`, `SubClassOf(C B)`, `SubClassOf(B A)`. Let `?X` be a class variable occurring in the axiom `SubClassOf(?X B)`. Then `?X` can be bound to either `A` or `C` and we get the bounded axioms `SubClassOf(A B)` or `SubClassOf(C B)`. Let `??X` be a multi-variable (range bounded to classes). Given the axiom `SubClassOf(??X B)`, `??X` would be bound to the set $\{A, B\}$ and we get the axioms `SubClassOf(A B)` and `SubClassOf(C B)`.

**Definition 3 (Constraints).** *Every OWL axiom and variablized axiom according to the Definition 2 is a constraint. Let* `?x` *and* `?y` *be variables, and* `??Z` *a multi-variable. Then the following statements are also constraints:* `?x != ?y`, `?x = ?y` *(in- resp. equivalence of variable bindings),* `MinCount(??Z) = n`, `MaxCount(??Z) = n`, *resp.* `ExactlyCount(??Z) = n` *(specifying a minimum, maximum or exact number of bounded variables* `?Z(0)`*,...,* `?Z(n)` *in* `??Z`*).*

**Definition 4 (Abstract OWL Pattern).**

```
IRI iri
<ACTION>
LET variables
WHERE EXPLICIT constraints
WHERE IMPLICIT constraints
TYPE type
MESSAGE message
DOCUMENTATION documentation
```

*is an abstract pattern where* iri *is a unique identifier,* $\langle ACTION \rangle$ *is an action's placeholder,* variables *is a comma-separated set of variables containing all variables that are used in the pattern, and* constraints *is a comma-separated set of (conjunctively connected) constraints according to Definition 3. Axioms mentioned in the* WHERE *statement are either interpreted as explicit or implicit axioms.* type *denotes the type of the pattern (as explained in the following),* documentation *is a human-readable documentation of the pattern, and* message *is a message that can be displayed in tools.*

Following the type categorization of patterns given in [4] and `ontologydesignpatterns.org` we use the given types also in our OWL patterns. Therefore we extended the meta-ontology about types with a new one for ontology lint patterns. It depends on the pattern's type which $\langle ACTION \rangle$ is provided. By default

```
ADD axioms
REMOVE axioms
```

where axioms is a set of axioms according to the Definition 2 are defined. For instance, ODPs typically only use the ADD statement whereas re-factoring patterns also provide a remove section, and lint patterns might inform the user only (they do not provide any action).

Ontology patterns are often based on either entities or axioms defined in other patterns: combining patterns or re-using patterns is a key building block of pattern creation. We therefore introduce the following statement group

```
FROM iris
AS constraints
```

where *iris* is a set of comma-separated pattern identifiers (IRIs) and *constraints* is a set of comma-separated constraints referring to variables in the patterns identified by their IRI and variables of the surrounded pattern. To distinguished structural equivalent variables from different patterns it is always required that these variables are prefixed with the IRI. All axioms and variables contained in the patterns mentioned in *iris* are always part of the surrounding patterns. This allows, for instance, to combine two patterns and establish equivalence between variables with help of the `AS` statement.

In the following some very short examples of using our additional language constructs are given. An ontology lint pattern that finds redundant transitive property axioms because of the transitivity of the super-property can be formulated as follows:

```
IRI http://www.derivo.de/patterns/WOP2009/RedundantTransitiveAxiom
LET ?X - ObjectProperty, ?Y - ObjectProperty
REMOVE TransitiveObjectProperty(?X)
WHERE EXPLICIT SubObjectProperty(?X ?Y),
               TransitveObjectProperty(?X),
               TransitiveObjectProperty(?Y)
```

A *cyclic class definition* can easily be defined by using variablized class expressions as follows:

```
IRI http://www.derivo.de/patterns/WOP2009/CyclicExplicitClasses
ADD EquivalentClasses ??X
REMOVE SubClassOf (??X ??Y) - ?X(i) = ?Y(i+1), ?X(0) = ?Y(n)
LET ?X - ClassExpression
WHERE EXPLICIT SubClassOf(??X ??Y) - ?X(i) = ?Y(i+1), ?X(0) = ?Y(n)
DOCUMENTATION Cyclic class definitions are replaced
  by an EquivalentClassAxiom
```

An ontology lint patterns discovering at least 5 super-classes (borrowed from Pellint's lint collection) can be expressed as follows:

```
IRI http://www.derivo.de/patterns/WOP2009/AtLeastFiveSuperClasses
LET ?X - ClassExpression, ?Y - ClassExpression
WHERE EXPLICIT SubClassOf(?X ??Y), MinCount(??Y) = 5
```

## 4 Implementation

We have integrated the proposed language constructs in a pattern language as a supplement to the OWL API [11]. In order to apply constraints on variable bindings (esp. in the case of multi-variables) we implemented methods known from CSP. For a pro-actively user-support during ontology engineering we have integrated the language within the successor of our graphical-based ontology authoring framework OntoTrack [12]. Here, the user can instantiate patterns by selecting them from catalogues and filling variables either by selecting corresponding OWL object or by drag-'n-drop them to the bound variable. Constraints of a pattern instantiation will be immediately checked and monitored during ontology engineering in order to avoid undesired constraint violations. For instance, Figure 4 shows a screen capture of the graphical ontology view within our application presenting the hierarchy of classes and properties in a geographical domain. Here, a class labeled `Partition` has been introduced by

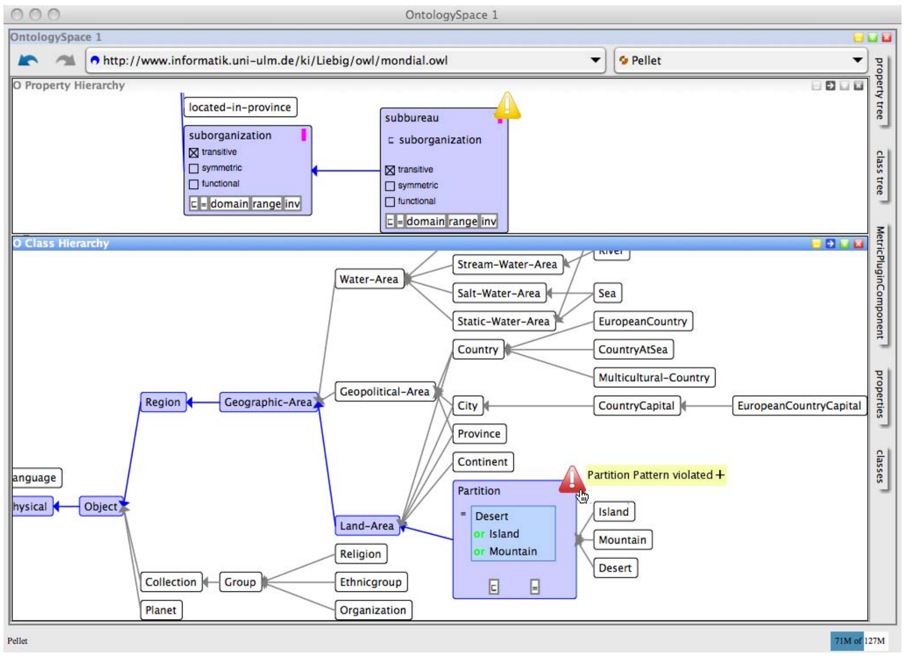

**Fig. 1.** Screen capture showing class hierarchy with pattern instantiation warnings in our OntoTrack-based ontology authoring framework.

applying the partition pattern. After removing the `DisjointClasses` axioms the user's intention of defining a partition is violated and therefore our framework immediately presents a hint to the user as shown as symbol in the right upper corner of the class box. Hovering over the class with the mouse pointer displays which pattern has been violated. Moreover, when clicking on the symbol one gets further information about the reason(s) for the violation.

Since ontology lint are handled in the same way as ODPs the same mechanism is used to monitor the ontology in order to find lints as a background task. As shown in the upper part of Figure 4, a redundant transitive axiom for a property that is a sub-property of a transitive property has been found.

## 5 Conclusion and Future Work

In this paper we presented some shortcomings of ontology pattern languages such as the OPPL-based ontology language and motivated to broaden the term ontology pattern by looking at ontology analysis methods which encode ontology patterns in program structures. A declarative pattern approach could benefit from automatic detection of pattern instantiations, monitoring of patterns violations as well as interoperability of patterns. To overcome the presented limitations we briefly introduced the building blocks of a universal pattern language and gave an overview of the implementation and integration in an ontology authoring environment. Future work will be concerned with further analysis of ontology patterns related areas such as ontology lints as well as ontology extractions to include language constructs to better support these patterns in order to get to our aim to build a universal pattern language.

## References

1. Fiedland, N., Allen, P., Witbrock, M., Matthews, G., Salay, N., Miraglia, P., Angele, J., Stab, S., Israel, D., Chaudhri, V., Porter, B., Barker, K., Clark, P.: Towards a Quantitative, Plattform-Independent Analysis of Knowledge Systems. In: Proc. of the 9th KR Conference, Whistler, BC, Canada (2004) 507–514
2. Rector, A.L., Drummond, N., Horridge, M., Rogers, J., Knublauch, H., Stevens, R., Wang, H., Wroe, C.: OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns. In: Proc. of the 14th Int. Conf. on Engineering Knowledge (EKAW). (2004) 63–81
3. Clark, P., Thompson, J., Porter, B.W.: Knowledge Patterns. In: Handbook on Ontologies. (2004) 191–208
4. Gangemi, A.: Ontology Design Patterns for Semantic Web Content. In: Proc. of 4th Int. Semantic Web Conference (ISWC 2005). (2005) 262–276
5. Presutti, V., Gangemi, A.: Content Ontology Design Patterns as Practical Building Blocks for Web Ontologies. In: Proc. of the 27th Int. Conf. on Conceptual Modeling (ER). (2008) 128–141
6. Presutti, V., Gangemi, A., del Carmen Suarez-Figueroa, M.: Library of Design Patterns for Collaborative Development of Networked Ontologies. Deliverable D.2.5.1 NeOn project (2007)
7. Iannone, L., Rector, A.L., Stevens, R.: Embedding Knowledge Patterns into OWL. In: Proc. of the 6th European Semantic Web Conference (ESWC 2009). 218–232
8. Motik, B., Patel-Schneider, P.F., Parsia, B.: OWL 2 Structural Specification and Functional-Style Syntax. W3C Candidate Recommendation 11 June 2009
9. Lin, H., Sirin, E.: Pellint – A Performance Lint Tool for Pellet. In: Proc. of the 5th OWLED Workshop on OWL: Experiences and Directions. (2008)
10. Kubias, A., Schenk, S., Staab, S., Pan, J.Z.: OWL SAIQL – An OWL DL Query Language for Ontology Extraction. In: Proc. of the OWLED 2007 Workshop on OWL: Experiences and Directions (OWLED'07). (2007)
11. Horridge, M., Bechhofer, S., Noppens, O.: Igniting the OWL 1.1 Touch Paper: The OWL API. In: Proc. of the OWLED 2007 Workshop on OWL: Experiences and Directions (OWLED'07). (2007)
12. Liebig, T., Noppens, O.: OntoTrack: A Semantic Approach for Ontology Authoring. Journal of Web Semantics **3**(2) (2005)