Making a Case for Formal Relations over Ontology Patterns

Daniel P. Lupp, Leif Harald Karlsen, and Martin G. Skjæveland

Department of Informatics, University of Oslo
{danielup,leifhka,martige}@ifi.uio.no

Abstract. There have recently been multiple frameworks proposed to formalize the definition and instantiation of recurring patterns for ontology construction and maintenance. Such formal frameworks can also provide the means necessary for discussing how such patterns can be related to one another, both syntactically and semantically. This has the potential for organizing pattern libraries, robust handling of maintenance tasks, such as redundancy removal, and defining heuristics for what constitutes a "good" pattern. This short paper aims to provide a common ground for discussions on formal relations between ontology patterns. We discuss interesting relations with motivating examples as well as state open questions concerning relations for optimizing the creation, instantiation, and maintenance of ontology patterns.

1 Templating Framework for Ontology Patterns

Recently, multiple frameworks have been proposed for the creation and use of ontology patterns [3,2]. This more formal approach enables the study of how patterns are related to one another in ways that permit automatic analysis and repair. In order to adequately discuss formally defined relationships between patterns, we employ the notion of a templating mechanism for patterns using the following generic definitions: An *(ontology) pattern* is a set of OWL axioms or RDF triples; a *templating framework* has the following characteristics, where we consider the first three mandatory and the remaining optional:

1. identifiable patterns, called *templates*;
2. declaration of fixed and variable template parameters;
3. precise instantiation of templates;
4. support for nested template definitions, i.e., templates defined using other templates;
5. typed parameters;
6. cardinality for parameters, e.g,. *optional* and *mandatory*;
7. inherited semantics of the underlying language.

The discussion is motivated by our work with *Reasonable Ontology Templates* (OTTR) [3,1] which implements these features. Figure 1 gives a schematic example of an OTTR template, for more details we refer to [3]. The OTTR template framework has been successfully verified for construction and maintenance tasks on Aibel's large-scale Material Master Data (MMD) ontology. There approximately 1000 templates were used to represent the spreadsheet formats created and populated by the project to capture the domain knowledge for generating an ontology of ca. 80,000 classes. Experimental analysis of these templates based on simple relations has revealed a considerable potential

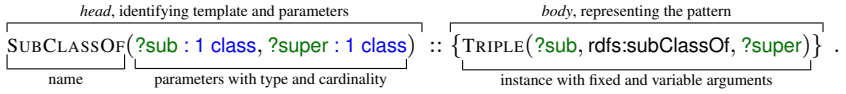

Fig. 1: Reasonable Ontology Templates (OTTR) exemplified.

for optimization of their design. We believe that a richer set of template relations is both possible and required in order to gain a more fine-grained and effective characterization of templates and template libraries.

In this paper we discuss the benefit and challenges of defining formal relations between ontology templates and identify future directions of research in this area, with an agnostic perspective as to which templating formalism is used. As such, the intention is for the reader to quickly be able to translate the discussion and examples given into the formalism of their choosing. Section 2 shows how simple relations may be used to identify redundancies in a template library. Section 3 presents the possibilities and potential for defining new relations over ontology templates.

2 Simple and Useful Template Relations

Using only the first four characteristics of a template framework given in Section 1, we can define some basic relationships between templates:

directly depends S is said to *directly depend on* T if S contains an instance of T in its definition.

depends *depends* is the transitive closure of *directly depends*.

dependency-overlaps S *dependency-overlaps* T if there exists a template upon which both S and T directly depend.

overlaps S *overlaps* T if there exist template instances i_S, i_T in the definition of S and T and substitutions ρ and η of the parameters of S and T resp. such that $\rho(i_S) = i_T$ and $\eta(i_T) = i_S$.

contains S *contains* T if there exists a substitution ρ of the parameters of T such that ρ applied to the pattern of T is a subset of the pattern of S.

These relations can be used to identify redundancies in a set of templates. In [3], two types of redundancies are considered: *lack of reuse* and *uncaptured pattern*. *Lack of reuse* occurs when a template duplicates the pattern of another template rather than instantiating it. This is exemplified by the two templates in Figure 2(a); BURGER1 *contains* SUBOBJECTALLVALUESFROM (see (1) and (2)). The template BURGER2 in Figure 2(b) is the result of fixing this redundancy (replacing (1) with (4)). *Uncaptured pattern* is the case when multiple templates make use of a pattern which is not represented by a template. This is illustrated by the two templates of Figure 2(b); these templates dependency-overlap, as seen in (3)–(4) and (5)–(6). This is not an occurrence of lack of reuse, as PIZZA cannot instantiate BURGER2 given they both use different fixed parameter resources (e.g., hasCondiments (4) and hasTopping (6)). The uncaptured pattern may be refactored out as a separate template; this is implemented by FOOD1 in Figure 2(c).

```
Burger1(?Name : 1 class, ?Condiments : + class)
  :: SubClassOf(?Name, :Burger), ObjectUnionOf(_:b3, ?Condiments),
     SubClassOf(?Name, _:b2), ObjectAllValuesFrom(_:b2, :hasCondiment, _:b3).   (1)

SubObjectAllValuesFrom(?x : 1 class, ?Property : 1 objectProperty, ?Range : 1 class)
  :: SubClassOf(?x, _:b1), ObjectAllValuesFrom(_:b1, ?Property, ?Range) .      (2)
```

(a) Example of lack of reuse.

```
Burger2(?Name : 1 class, ?Condiments : + class)
  :: SubClassOf(?Name, :Burger), ObjectUnionOf(_:b1, ?Condiments),             (3)
     SubObjectAllValuesFrom(?Name, :hasCondiment, _:b1).                       (4)

Pizza(?Name : 1 class, ?Toppings : + class, ?Country : ? individual)
  :: SubClassOf(?Name, :NamedPizza), ObjectUnionOf(_:b1, ?Toppings),          (5)
     SubObjectAllValuesFrom(?Name, :hasTopping, _:b1),                         (6)
     SubObjectHasValue(?Name, :hasCountryOfOrigin, ?Country).
```

(b) Example of uncaptured pattern.

```
Food1(?Name : 1 class, ?Type : 1 class, ?Ex : + class, ?hasEx : 1 objProp)
  :: SubClassOf(?Name, ?:Type), ObjectUnionOf(_:b1, ?Ex),
     SubObjectAllValuesFrom(?Name, ?:hasEx, _:b1).
```

(c) Captured pattern.

```
Food2(?Name : 1 class, ?Type : 1 class, ?Ex : + class, ?hasEx : 1 objProp, ?Country : ? individual)
  :: SubClassOf(?Name, ?:Type), ObjectUnionOf(_:b1, ?Ex),
     SubObjectAllValuesFrom(?Name, ?:hasEx, _:b1),
     SubObjectHasValue(?Name, :hasCountryOfOrigin, ?Country).
```

(d) Captured pattern with optionals.

Fig. 2: Templates demonstrating different redundancies and suggested solutions.

In the analysis of the ∼1000 templates used for Aibel's MMD ontology, ca. 55 million potential redundancies were identified and 367 possibly superfluous templates found. When deciding how to refactor the template library we used a manual approach, targeting large templates that model specific domain facts. Fixing a single redundancy reduced the total number of potential redundancies by more than 1.8 million. Although the analysis is clearly useful and shows promising results, we believe these large figures show that one has yet to find precise, effective means to characterize and repair template libraries.

3 Defining New Ontology Template Relations

The relations defined in the previous section use only a few of the basic characteristics for a templating mechanism given in Section 1. The following is a non-exhaustive

list of building blocks for defining formal relations between templates using both the characteristics of a templating framework and of the underlying language. The list is presented along with a brief summary of what kind of functionality each provides and typical examples of possible relations defined using these characteristics:

Syntax and semantics of framework These allow syntactic and semantic relations such as containment, entailment, and consistency, e.g., two templates could be said to be *inconsistent* if for any substitutions of their parameters the use of both in conjunction yields an inconsistent ontology.

Parameters This allows for relations that take into consideration the number of parameters, their types, and whether they are optional or not. For instance, one could say that T *strengthens the typing* of S if T directly depends on S and additionally requires a more specific type for at least one of its parameters.

Expansion This allows for relations over both unexpanded and expanded templates. For example, *expanded containment* could be defined to hold between T and S if the expansion of T contains the expansion of S.

Metrics Characteristics such as the template's *arity* (the number of parameters), *width* (the size of its pattern) and *depth* (the height of its expansion tree).

Syntax and semantics of used vocabulary It is useful to distinguish between templates that are completely generic (all parameters are variable) and those that use a specific vocabulary, be it a "logical" vocabulary like RDF or OWL or a more special purpose vocabulary. As for semantic relations such as consistency, these can be set to consider or ignore the semantics of "external" ontologies.

These building blocks give many possibilities for characterizing templates and template libraries. We are particularly interested in identifying relations that are useful for structuring template libraries for maintenance tasks, such as redundancy removal, and that make it easier for the user to find the relevant template for the modeling task at hand. In addition, finding heuristics with which libraries can be optimized according to specific metrics, which may vary from case to case, could be a valuable asset. Ideally, the relations and heuristics should be efficiently computable and be easily understood by the user. Finding these requires more theoretical and empirical study. We conclude this section by demonstrating some of the challenges of such studies.

When defining relations using the above building blocks, one should proceed with caution as some of them are incredibly powerful. For instance, allowing parameters to be optional provides more possibilities for discovering candidates for uncaptured patterns, as demonstrated in Figure 2: Consider the PIZZA and BURGER2 templates from Figure 2(b). Without allowing optional parameters, a suitable template that describes the uncaptured pattern can be seen in Figure 2(c). With optionals, however, the BURGER2 and PIZZA templates can be generalized into a single template by declaring ?Country as an optional parameter, seen in Figure 2(d). Optionals thus provide a lot of useful functionality; however, in general they allow for any set of templates to be summarized into a single template. As this is likely counter-productive in most circumstances, it would be important to identify heuristics for when optionals should or should not be used when defining templates for uncaptured patterns.

Other complex relationships can occur during the fixing of lack of reuse or uncaptured pattern. An example that illustrates this is shown in Figure 3, where fixing one instance

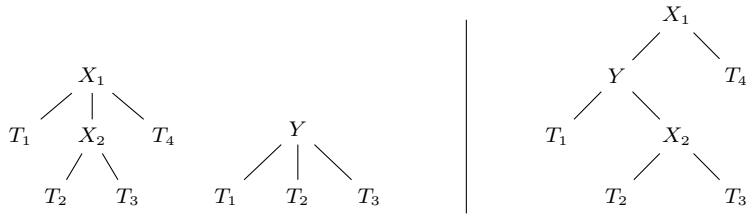

Fig. 3: A complex case of lack of reuse. Templates are schematically represented by trees where an edge represents a direct dependency. On the left, there is a lack of reuse between the templates X_2 and Y, which, if fixed (by replacing T_2 and T_3 in Y with X_2), introduces a new lack of reuse between Y and X_1. Fixing also this redundancy (replacing T_1 and X_2 in X_1 with Y) results in the configuration of templates seen on the right.

of lack of reuse in Y introduces a new lack of reuse in X_1. This indicates a more complex form of lack of reuse between X_1 and Y in the original library, but also, more generally, that repairing redundancies is an iterative process. This type of composition of relations opens the door to new types of complex relationships. Furthermore, if the template definitions have type and cardinality restrictions set on the parameters, finding possible optimizations or redundancies becomes even more involved: for instance, if parameter types are too strict then the possible reuse of these templates is reduced.

4 Conclusion and Future Work

In this paper, we introduce an abstract templating framework and the building blocks with which formal relations over ontology patterns and templates may be defined. We give examples of formal relations that have a demonstrable benefit to template library maintenance (by removing redundancy) as well as provide useful functionality for tools geared towards template creation, documentation, and discovery.

The framework, relations and building blocks described in this paper serve primarily as a basis for discussion. We believe it important to identify (1) what functionality we desire for improved usability and in maintenance and creation tools, (2) what relations these require, and (3) what properties aside from the aforementioned building blocks a templating framework should support.

References

1. H. Forssell, D. P. Lupp, M. G. Skjæveland, and E. Thorstensen. Reasonable macros for ontology construction and maintenance. In *Proc. of 30th DL Workshop*, 2017.
2. B. Krieg-Brückner and T. Mossakowski. Generic ontologies and generic ontology design patterns. In *Proc. of the 8th Workshop on Ontology Design and Patterns*, 2017.
3. M. G. Skjæveland, D. P. Lupp, L. H. Karlsen, and H. Forssell. Practical ontology pattern instantiation, discovery, and maintanence with reasonable ontology templates. Accepted for ISWC 2018 research track, 2018.