Practical Ontology Pattern Instantiation, Discovery, and Maintanence with Reasonable Ontology Templates

Demo paper

Martin G. Skjæveland, Leif Harald Karlsen, and Daniel Lupp
{martige,leifhka,danielup}@ifi.uio.no

Department of Informatics, University of Oslo

Abstract. Reasonable Ontology Templates (OTTR) is a language for representing ontology modelling patterns in the form of parameterised ontologies. Ontology templates are simple and powerful abstractions useful for constructing, interacting with and maintaining ontologies. The templates' simple yet formally precise structure allows for formal relations to be defined over templates, supporting sophisticated maintenance tasks for template libraries such as revealing redundancies and suggesting new templates for representing uncaptured modelling patterns.

1 Introduction

Constructing sustainable large-scale ontologies of high quality is hard. Part of the problem is the lack of established tool-supported best practices for ontology construction and maintenance. *Reasonable Ontology Templates* (OTTR) provide a simple but powerful abstraction mechanism based on the well-known concept of nested non-cyclic macros and syntactic substitutions. This allows complex ontology expressions to be compactly represented by a naturally compositional structure which we believe supports more efficient construction and maintenance of ontologies following "don't repeat yourself" (DRY) principles. Furthermore, templates formally represent parameterised ontologies and can be compactly represented in RDF allowing us to leverage the stack of existing W3C languages and tools. This paper demonstrates a prototype implementation of the results presented in [1].

2 Reasonable Ontology Templates

An *OTTR template* T consists of a *head*, specifying the template's *name* and *parameters*, and a *body*, representing a parameterised ontology pattern. A *template instance* consists of a template name and a list of *arguments* that matches the parameters of the designated template, and represents a replica of the template's body pattern where parameters are replaced by the instance's arguments. Each template parameter has a *type* and a *cardinality*, with which the permissible types and number of arguments a parameter accepts are specified. The template body comprises only template instances, i.e., the template pattern is recursively built up from other templates—cyclic template dependencies are not allowed. There is one special *base template*, TRIPLE, which takes three arguments and

Fig. 1. The NamedPizza template (top), instances of the template (middle), and the instances expanded (bottom).

has no body, but represents a single RDF triple in the obvious way. *Expanding* an instance is the process of recursively replacing instances with the pattern they represent. Template instances may be expanded under a *mode* which allows multiple instances of the template to be generated for arguments which are lists. The expansion process terminates with an expression containing only Triple template instances, hence representing an RDF graph.

Different serialisation formats of OTTR templates exist, including an RDF serialisation supported by a special-purpose OWL vocabulary and a language for representing template instances in tabular formats such as spreadsheets.

Example 1. Figure 1 shows the NamedPizza template, example instances of the template, and the result of expanding these instances. The head and body of the template are separated by '::' . The body contains instances that represent common OWL axioms. The head specifies three parameters, ?Name, ?Country, ?Toppings, respectively with the types class, individual, and class and the cardinalities 1 (*mandatory*), ? (*optional*), and + (*multiple*). The type of the second parameter (individual) requires its argument to be an OWL individual and its cardinality (*optional*) allows a null-value, denoted none. The effect of using null-value arguments is illustrated with the expansion of the Grandiosa instance which does not include a value restriction on its country of origin due to the second argument being none. The SubObjectSomeValuesFrom instance in the template body is marked with an expansion mode, denoted by x. Its effect is that the expansion generates one instance of the template per item in the ?Toppings list argument. This is seen in the expansions by the multiple occurrences of existential restriction axioms on hasTopping. Notice that the toppings list is also used to form a union of toppings which qualifies the universal restriction axiom. The online library listing of this template at `http://osl.ottr.xyz/info/?tpl=http://draft.ottr.xyz/pizza/NamedPizza` displays its RDF serialisation, including visualisations and queries generated from the template and links to dependant templates.

Fig. 2. OTTR template library before (top) and after (bottom) redundancy removal. Numbers indicate redundancies, a dagger (†) marks a change to an existing template and a star (⋆) marks an added template.

3 Maintenance of Template Libraries

Automatic detection of redundancy in OTTR template libraries is made possible by OTTR's formal foundation and simple structure. In this section, we present two types of redundancy: (i) a lack of reuse of existing templates, and (ii) recurring patterns not captured by templates within the library. If the body of a template T is a subset of another template R's body, assuming an appropriate substitution of the variables of T, then R has a *lack of reuse* of T. This can be fixed by simply substituting the offending instances in R by an appropriate instance of T. A case of *uncaptured pattern* is when a pattern of template instances occur across multiple templates without there being any template with this pattern as its body. To fix this type of redundancy, we first have to introduce a new template with the recurring pattern as its body and then substitute the pattern in the offending templates by an appropriate instance of this newly introduced template. In Fig. 2, an example library containing redundancies is presented, together with a library where the redundancies have been fixed. A lack of reuse (of ANNOTATION) is marked with (1), and an uncaptured pattern (also used by NAMEDPIZZA) is marked with (2).

A naive method for finding the two types of redundancy based on direct unification of subsets of the template's bodies is infeasible for large template libraries. We have therefore developed an efficient method for finding lack of reuse and uncaptured patterns, which over-approximates the results of unification based on the notion of a dependency pair. A *dependency pair* is a pair $\langle I, T \rangle$ of a multiset of templates I and a set of templates T, such that all templates in T have at least as many occurrences of each template in I as they occur in I in its body. The idea is that I describes a possible pattern used by the

templates T, but without considering unification of parameters. In order to also detect patterns containing different TRIPLE instances, we treat a TRIPLE instance (s, p, o) as an instance of the form $p(s, o)$ and thus the predicate p as a template.

One can compute all dependency pairs by starting with the set of dependency pairs of the form $\langle \{i : n\}, T \rangle$ where all templates in T have at least n instances of i, and then compute all possible *merges*, where a merge between $\langle I_1, T_1 \rangle$ and $\langle I_2, T_2 \rangle$ is $\langle I_1 \cup I_2, T_1 \cap T_2 \rangle$. We have implemented such an algorithm with a few optimisations that ensures each dependency pair is computed only once.

For each template $t \in T$ in each dependency pair $\langle I, T \rangle$ we can construct a new template s by extracting the instances in t's body corresponding to the instances in I, with all parameters and constants occurring in this set of instances as parameters. For each such template s there are three cases: (i) s does not unify with the corresponding instances in any other template in T and hence does not represent a repeated pattern; (ii) s is equal to an already existing template, in which case we have found an instance of *lack of reuse* in the other templates of T which share the pattern; (iii) s is an uncaptured pattern which can be introduced to the library. Refactoring should always be supervised; suggested templates need not represent natural modelling patterns, and selected templates must be named and their parameters given an appropriate name and type.

Example 2. The redundancies from Fig. 2 are captured by two dependency pairs:

⟨ {rdfs:label, skos:prefLabel, skos:definition}, {ANNOTATION, BURGER} ⟩

⟨ {SUBCLASSOF, SUBOBJECTSOMEVALUESFROM, SUBOBJECTALLVALUESFROM, OBJECTUNIONOF},
{NAMEDPIZZA, BURGER } ⟩

By extracting the occurrences of these patterns as described above, we get two template suggestions:

⟨NAME⟩(?x1, ?x2, ?x3, ?x4) :: (?x1, rdfs:label, ?x2), (?x1, skos:prefLabel, ?x3), (?x1, skos:definition, ?x4) .

⟨NAME⟩(?x1, ?x2, ?x3, ?x4) :: SUBCLASSOF(?x1, ?x2), x | SUBOBJECTSOMEVALUESFROM(?x1, ?x3, ?x4),
SUBOBJECTALLVALUESFROM(?x1, ?x3, _:b4), OBJECTUNIONOF(_:b4, ?x4) .

The first is equal to the ANNOTATION template, and shows a lack of reuse in BURGER. The second is not equal to any template, and is introduced as a new template NAMEDFOOD.

4 Demonstration

An executable demonstration based on our prototype implementation can be found at https://www.ottr.xyz/event/2018-10-08-iswc/. The demonstration gives more examples of OTTR templates, shows the expansion of template instances, demonstrates different serialisations of templates and instances, and applies the automatic redundancy detector for template library maintenance showing dependency pairs and corresponding template suggestions for uncaptured patterns.

References

1. 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 ISWC2018 research track, 2018.