# Foundations of Ontology Template Language OTTR (Extended Abstract)

Erik Snilsberg[1], Leif Harald Karlsen[1], Egor V. Kostylev[1] and Martin G. Skjæveland[1]

[1]*Department of Informatics, University of Oslo, Norway*

**Abstract**

OTTR is a template language for the Semantic Web data and knowledge representation languages RDF and OWL. OTTR sees use as a key technology in the Semantic Web systems and already gained popularity both in academia and industry. OTTR is also in active development, which is, however, limited by the fact that little is known about its theoretical aspects. In this extended abstract we begin to close this gap by laying the foundations of OTTR. To this end, we first introduce a mathematical formalisation of the language in terms of its syntax and semantics. Then, we report bounds for the size of OTTR expansion—that is, of the data encoded by templates. Finally, we overview our study of the data and combined complexity of an important decision problem associated with the language and its fragments.

**Keywords**

OTTR, Semantic Web, ontology management, ontology templates, theoretical foundations

## 1. Introduction

The Resource Description Framework (RDF) [1] and the Web Ontology Language (OWL) [2] are among the central knowledge representation languages for practical use. RDF is a generic data model where statements are expressed as triples, thus giving RDF datasets a graph-like structure, favourable for many applications. OWL is an ontology language based on description logics [3], which ensure desirable computational properties. OWL can be serialised in RDF by means of a simple mapping [4]. Both languages are standardised (see https://www.w3.org/RDF/, https://www.w3.org/OWL/) and supported by a wide range of software. Although RDF and OWL are popular, engineering and maintaining such knowledge bases are not trivial [5]. This is caused in part by the lack of support for user-defined abstraction, which forces all expressions to be made as RDF triples and OWL axioms, which often appears too low-level for users.

Reasonable Ontology Templates (OTTR) [6] is a recent template language designed to increase the efficiency and quality of constructing and maintaining RDF and OWL knowledge bases. The utility of OTTR comes from its ability to represent and instantiate user-defined parameterised modelling patterns. OTTR allows to introduce sound modelling principles to knowledge base engineering that are not well-supported by established semantic technologies, such as encapsulation of modelling complexity, modelling uniformity, separation of concerns, and avoiding repetitions. OTTR can thereby provide actionable resources that promote correct use of the vocabulary—similar to what programming APIs provide to software engineers [7].

In a nutshell, an OTTR *dataset* consists of templates and instances. An OTTR *template* represents a parameterised encoding of a (knowledge base) modelling pattern, and an *instance* of a template represents a replica of the pattern where the template's parameters are substituted by the instance's arguments. Templates form a (non-cyclic) recursive structure; the pattern of a template is specified by a

*DL 2024: 37th International Workshop on Description Logics, June 18–21, 2024, Bergen, Norway*

eriksni@ifi.uio.no (E. Snilsberg); leifhka@ifi.uio.no (L. H. Karlsen); egork@ifi.uio.no (E. V. Kostylev); martige@ifi.uio.no (M. G. Skjæveland)

https://www.mn.uio.no/ifi/english/people/aca/eriksni/ (E. Snilsberg); http://leifhka.org (L. H. Karlsen); https://www.mn.uio.no/ifi/english/people/aca/egork/ (E. V. Kostylev); https://www.mn.uio.no/ifi/english/people/aca/martige/ (M. G. Skjæveland)

0009-0005-3039-3778 (E. Snilsberg); 0000-0001-5131-5246 (L. H. Karlsen); 0000-0002-8886-6129 (E. V. Kostylev); 0000-0002-9736-8316 (M. G. Skjæveland)

set of template instances which may use template parameters as arguments. *Base templates* are used to directly represent structures in the base language (e.g., RDF). A set of instances can be translated to the base language by means of (template) *expansion*, which recursively replaces instances with their corresponding template replicas until only instances of base templates are left. To have a feeling for OTTR, let us consider a dataset with two templates, with names ex:Person and o-rdf:Type, taking two parameters each:

```
ex:Person[?person, ?name] :: {
  o-rdf:Type(?person, foaf:Person), ottr:Triple(?person, foaf:name, ?name)} .
o-rdf:Type[?instance, ?class] :: {ottr:Triple(?instance, rdf:type, ?class)} .
```

Note that ex:Person relies on o-rdf:Type and another template, ottr:Triple, which is a base template whose instances represent RDF triples. The dataset also has an instance ex:Person(:Bob, "Bobby Jackson"), which expands in two steps to two RDF triples: (:Bob, rdf:type, foaf:Person) and (:Bob, foaf:name, "Bobby Jackson"). This example illustrates only the basic OTTR functionality. However, OTTR has more advanced features, including special treatment for possibly nested lists and sophisticated list expansion, which we illustrate later.

An OTTR specification, as well as its serialisation formats, are available as technical reports [8, 9, 10]; there is an open-source reference implementation of OTTR, *Lutra* (http://www.ottr.xyz). OTTR is in active development and is used by a wide range of projects [11, 12, 13]. There are several approaches with a purpose similar to the one of OTTR [14, 15, 16, 17, 18]. However, OTTR has features that are not present in the other systems, most prominently, list expansion.

While OTTR has been gaining in popularity, its development is limited by the fact that it lacks a mathematical formalisation and that little is known about its theoretical properties. In this extended abstract, we report that we begin to close this gap and lay down the foundations of OTTR. In particular, we first introduce a mathematical formalisation of a core of the OTTR language, including its key advanced features, thus enabling its formal studies; we then report several upper bounds on the size of the result of OTTR expansion, giving an indication of how concisely OTTR can encode ontologies; finally, we overview our study combined and data computational complexity of OTTR expansion and establish fine-grained borders between OTTR fragments, separating those that can be easily evaluated from those that are more difficult.

## 2. Key Features of OTTR

The key features covered in our formalisation are optional parameters, default values, and list expansion modes (the latter of which is the most interesting and demanding from the theoretical point of view), and we introduce them in this section by means of examples. Note, however, that OTTR specifications [8, 19] include additional features, such as type system for parameters, non-blank parameters, which are omitted from our formalisation for brevity.

We begin with our first two features. As illustrated by the following example, parameters marked as optional with symbol ? allow a special term none to be used as an argument for these parameters. Meanwhile, for non-optional parameters, if none is given as an argument in some instance, then the entire instance is ignored during expansion. A parameter may also have a default value, specified as a predefined constant. When none is given as an argument for a parameter with a default value, it is replaced by the default value.

For example, we can extend the ex:Person template of the example in the introduction to include information about the email address and the location of the person:

```
ex:Person[?person, ?name, ? ?mail, ?loc=ex:Norway] :: {
  o-rdf:Type(?person, foaf:Person), ottr:Triple(?person, foaf:name, ?name),
ottr:Triple(?person, foaf:mbox, ?mail), ottr:Triple(?person, foaf:based, ?loc)} .
```

Since parameter ?mail is marked by (another) ? as optional, we may give none as the corresponding argument. In that case, all occurrences of ?mail are replaced with none in the ex:Person template during its expansion. However, since the ottr:Triple template is non-optional for all parameters, the second instance of ottr:Triple would be ignored. Next, since ?location has a default value, we can also give none as the corresponding argument. In this case, ?location uses its default value instead. So, overall, expanding the instance ex:Person(:Bob, "Bobby Jackson", none, none) results in the triple (:Bob, foaf:based, ex:Norway), in addition to the two triples generated in the original example.

We move on to the third key feature, list expansions, which is used to access the elements of list terms. There are three expansion modes: *cross*, *zipMin*, and *zipMax*. Instances can be specified as having one of these modes, indicating special treatment for list arguments during expansion. Only arguments marked with ++ are affected, while unmarked arguments are treated as normal, even if they are lists. An error is raised if a non-list argument is marked with ++. For all three expansion modes, new instances are created during expansion, with the elements of the lists replacing the lists themselves as instance arguments. In particular, cross combines the list elements via the Cartesian product of the input lists, while zipMin and zipMax combines elements that have the same position in the input lists. The difference between the latter two is that zipMin combines elements up the length of the shortest list, while zipMax pads shorter lists with none up to the length of the longest list.

For example, given two lists of people (:Bob, :John, :Jane) and (:Stephanie, :Ross) we can specify which people in the first list knows the people in the second list. If each person in the first list knows everyone in the second list, then we can use expansion mode cross:

```
ex:Knows[?knowers, ?knowees] :: {
  cross | ottr:Triple(++ ?knowers, foaf:knows, ++ ?knowees) } .
```

Then, expanding the instance ex:Knows((:Bob, :John, :Jane), (:Steph, :Ross)) results in the six triples with all possible combinations of list elements, from (:Bob, foaf:knows, :Steph) to (:Jane, foaf:knows, :Ross). If instead the knowing-relation corresponds with position in the two lists (i.e., :Bob knows :Steph, :John knows :Ross, etc.), then we can use zipping modes zipMin or zipMax instead of cross. For example, replacing cross with zipMin results in the two triples (:Bob, foaf:knows, :Steph) and (:John, foaf:knows, :Ross). Since position of person :Jane exceeds the length of the shorter list, this element is simply ignored with zipMin. Replacing zipMin with zipMax and expanding the same instance results in the additional intermediate instance ottr:Triple(:Jane, foaf:knows, none), which is, however, ignored since the third parameter of ottr:Triple is non-optional, resulting in the same triples as for zipMin.

Finally, we bring attention to the fact that lists inside templates may have variables as elements, as in, for example, (:Bob, :John, ?person, :Jane). This is a way to construct new terms during expansion, which has a significant impact on the expressivity and complexity or OTTR.

## 3. Overview of Results

We first report several upper bounds for the size of expansion with respect to the size of datasets, which gives us an indication of the conciseness of OTTR. We established a general bound for full OTTR, as well as bounds for two of its important fragments: a fragment without list expansion and a fragment where all lists are ground (i.e., variable-free). In particular, the size of expansion is bounded exponentially by both template nesting depth and by maximal template arity. If no list expansion is allowed, then only template nesting is in the exponent (in fact, only cross mode is essential); if all lists are ground, then only template arity is in the exponent.

We next report our studies of the computational (combined) complexity of a central decision problem associated with the expansion process, namely, the problem Expansion of determining whether a given instance occurs in the expansion of a given error-free OTTR dataset. Studying Expansion helps us understand how difficult it is to retrieve the data stored in OTTR datasets. We consider Expansion for different syntactic fragments of OTTR, which allows us to separate the types of datasets which are easy to expand from the ones that are more difficult. In particular, we established a fine border, separating fragments of OTTR for which the combined complexity of Expansion is NP-complete and for which it is NL-complete. It is worth to note that the former includes the full OTTR, while the latter includes essentially the simplest possible fragment.

Finally, we report our results on data complexity of Expansion. In this setting, we assume that the templates of an OTTR dataset are 'fixed,' and only the instances of the dataset are considered to form the input. This gives a more practical view of the complexity of Expansion, because the number of templates is usually rather small, while the number of instances may be arbitrarily big. In this case, all relevant OTTR fragments fall into two complexity classes, $TC^0$-complete (under $AC^0$ reductions) and $AC^0$, and the separating property is the presence of lists and expansion modes zipMin or zipMax.

## 4. Conclusion and Future Work

In this extended abstract we report our results on the foundations of OTTR. This is just the beginning of OTTR formal studies, which opens up many further research avenues: we plan to formalise remaining OTTR features, including types, look at DL ontologies produced by OTTR, and study OTTR under a generalisation of OBDA. We also plan to investigate how our algorithms can be used to improve the efficiency of real-life OTTR systems, such as Lutra.

## References

[1] R. Cyganiak, D. Wood, M. Lanthaler, RDF 1.1 Concepts and Abstract Syntax, Technical Report, W3C, 2014.

[2] B. Motik, P. F. Patel-Schneider, B. Parsia, OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax, Technical Report, W3C, 2012.

[3] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press, 2003.

[4] P. F. Patel-Schneider, B. Motik, OWL 2 Web Ontology Language Mapping to RDF Graphs, Technical Report, W3C, 2012.

[5] T. Tudorache, Ontology engineering: Current state, challenges, and future directions, Semantic Web 11 (2020) 125–138.

[6] M. G. Skjæveland, D. P. Lupp, L. H. Karlsen, J. W. Klüwer, OTTR: Formal Templates for Pattern-Based Ontology Engineering, in: Workshop on Ontology Design and Patterns (WOP), volume 51, 2021, pp. 349–377.

[7] M. G. Skjæveland, H. Forssell, J. W. Klüwer, D. P. Lupp, E. Thorstensen, A. Waaler, Reasonable Ontology Templates: APIs for OWL, in: ISWC Posters&Demonstrations and Industry Tracks, 2017.

[8] L. H. Karlsen, M. G. Skjæveland, Concepts and Abstract Model for Reasonable Ontology Templates (mOTTR), Technical Report, 2023.

[9] M. G. Skjæveland, Web Reasonable Ontology Templates (wOTTR), Technical Report, 2023.

[10] M. G. Skjæveland, L. H. Karlsen, Terse Syntax for Reasonable Ontology Templates (stOTTR), Technical Report, 2023.

[11] M. Hodkiewicz, J. W. Klüwer, C. Woods, T. Smoker, E. Low, An ontology for reasoning over engineering textual data stored in fmea spreadsheet tables, Computers in Industry 131 (2021) 103496.

[12] Y. Svetashova, B. Zhou, T. Pychynski, S. Schmidt, Y. Sure-Vetter, R. Mikut, E. Kharlamov, Ontology-Enhanced Machine Learning: a Bosch Use Case of Welding Quality Monitoring, in: International Semantic Web Conference (ISWC), 2020, pp. 531–550.

[13] A. Ekelhart, F. J. Ekaputra, E. Kiesling, Automated Knowledge Graph Construction from Raw Log Data, in: ISWC Posters&Demonstrations and Industry Tracks, 2020.

[14] D. Vrandečić, Explicit Knowledge Engineering Patterns with Macros, in: Ontology Patterns for the Semantic Web Workshop, 2005.

[15] B. Krieg-Brückner, T. Mossakowski, Generic Ontologies and Generic Ontology Design Patterns, in: Workshop on Ontology Design and Patterns, 2017.

[16] L. Iannone, A. L. Rector, R. Stevens, Embedding Knowledge Patterns into OWL, in: Extended Semantic Web Conference (ESWC), 2009, pp. 218–232.

[17] M. Lefrançois, A. Zimmermann, N. Bakerally, A SPARQL Extension for Generating RDF from Heterogeneous Formats, in: Extended Semantic Web Conference (ESWC), 2017.

[18] P. Lord, The Semantic Web takes Wing: Programming Ontologies with Tawny-OWL, in: International Workshop on OWL: Experiences and Directions (OWLED), 2013.

[19] L. H. Karlsen, M. G. Skjæveland, Adapting Reasonable Ontology Templates to RDF (rOTTR), Technical Report, 2019.