OntoBench: Generating Custom OWL 2 Benchmark Ontologies

Vincent Link[1], Steffen Lohmann[2(✉)], and Florian Haag[1]

[1] Institute for Visualization and Interactive Systems, University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany
[2] Fraunhofer Institute for Intelligent Analysis and Information Systems (IAIS), Schloss Birlinghoven, 53757 Sankt Augustin, Germany
steffen.lohmann@iais.fraunhofer.de

Abstract. A variety of tools for visualizing, editing, validating, and documenting OWL ontologies have been developed in the last couple of years. The OWL coverage and conformance of these tools usually needs to be tested during development for evaluation and comparison purposes. However, in particular for the testing of special OWL concepts and concept combinations, it can be tedious to find suitable ontologies and test cases. We have developed OntoBench, a generator for OWL benchmark ontologies that can be used to test and compare ontology tools. In contrast to existing OWL benchmarks, OntoBench does not focus on scalability and performance but OWL coverage and concept combinations. Consistent benchmark ontologies are dynamically generated based on any combination of OWL 2 language constructs selected in a graphical user interface. OntoBench is available on GitHub and as a public service, making it easy to use the tool to generate custom benchmark ontologies and ontology fragments.

Keywords: Ontology · Benchmark · Generator · OWL 2 · Coverage · Conformance

1 Introduction

A large number of tools that support the visualization, editing, validation, and documentation of OWL ontologies have been developed in the last couple of years. During the development of such tools, it is important to test them with ontologies representing different test cases (henceforth called *benchmark ontologies*) in order to ensure that the language constructs of OWL are adequately represented. Benchmark ontologies are also useful to support the comparison and evaluation of existing tools in order to assess the features of the tools and to check whether they provide adequate support for a certain use case.

In our previous work [7,8], we developed a static benchmark ontology for the purpose of testing feature completeness of ontology visualization tools. With OntoBench, we took this idea one step further and made it generally applicable in different use cases. Many ontology tools do not aim for a complete coverage

P. Groth et al. (Eds.): ISWC 2016, Part II, LNCS 9982, pp. 122–130, 2016.
DOI: 10.1007/978-3-319-46547-0_13

of OWL but focus on specific aspects, or are designed to cover only some of the OWL profiles. To overcome the inflexibility and overhead caused by a static benchmark ontology, we developed a systematic approach to dynamically generate benchmark ontologies tailored to the OWL coverage and feature set that a tool intends to support.

As opposed to most other ontology benchmarks, OntoBench is not meant for testing the scalability or performance in terms of the number of elements contained in an ontology, but it rather aims to test the scope of ontology tools in terms of supported features and OWL constructs. Accordingly, it focuses on the representation of the TBox of ontologies (i.e., the classes, properties, datatypes, and a few key individuals), while it does not support the testing of ABox information (i.e., larger collections of individuals and data values), which is the focus of most of the related work.

2 Related Work

One well-known ontology benchmark is the Lehigh University Benchmark (LUBM) [6], a test suite for ontology-based systems. LUBM extends benchmarks for databases with a focus on the Semantic Web. It contains an ontology describing the university domain, a tool that generates instance data for the university ontology, and test queries for the data whose performance can be evaluated by using a couple of metrics provided by LUBM. Wang et al. extended the LUBM benchmark by implementing a domain-agnostic generator for instance data [18]. It uses a probabilistic model to generate a user-given number of instances based on representative data from the domain in focus. This enables testing a broader range of possible topics and, consequently, different kinds of ontology structures. As an example, they created the Lehigh BibTeX Benchmark (LBBM) on the basis of a BibTeX ontology.

Another extension of LUBM has been proposed by Ma et al. with the University Ontology Benchmark (UOBM) [12]. UOBM aims to contain the complete set of OWL 1 language constructs and defines two ontologies, one being compliant with OWL 1 Lite and the other with OWL 1 DL. In addition, several links were added between the generated instances in order to create more realistic data.

The W3C Ontology Working Group also published a number of small ontologies and ontology fragments together with the specifications of OWL 1 and 2, providing normative test cases [5,16]. These test cases are intended for validating applications in terms of conformance to the respective OWL version and to demonstrate the correct usage of OWL. The majority of these test cases aim at testing different syntaxes, specific combinations of OWL constructs, or reasoners.

Furthermore, there are benchmarks addressing certain aspects of ontology engineering. For instance, a number of datasets and test cases has been created in the context of the Ontology Alignment Evaluation Initiative (OAEI) [1]. They are intended to evaluate and compare the quality and performance of ontology matching methods in particular. Finally, benchmarks for comparing the performance of SPARQL endpoints are available that feature RDF data generators and provide sets of benchmark queries [4,15].

To conclude, there is currently no benchmark—except for the static OntoViBe ontology we developed in our previous work [7,8]—that focuses on testing ontology tools for feature-completeness with regard to OWL coverage, and which supports a major part of the concepts specified in OWL 2. For generating custom ontologies, one could use ontology editors like Protégé to manually create a benchmark ontology tailored to one's needs. However, despite the modeling support provided by ontology editors, reliably covering all relevant test cases can still be very error-prone and tedious for users.

3 Requirements and Design Considerations

To fill this gap, we developed OntoBench, a generator for benchmark ontologies with a focus on testing the OWL 2 coverage of ontology tools. The OWL language constructs contained in the generated ontologies are selected by the user. For this purpose, we have defined abstract features that encompass one or more OWL language constructs and thereby form a test case. The features can be individually enabled or disabled by the user when generating the benchmark ontology.

3.1 Requirements

The features defined for OntoBench were drawn from two main considerations:

1. A complete (as far as possible) coverage of OWL 2 language constructs was to be achieved, hence features were built around the list of language constructs.
2. OWL includes some concepts that cannot be represented by single language constructs but require combinations of constructs. Such combinations were also included as test cases.

Each test case can be seen as a fragment of the ontology to be generated. In order to optimally embed the fragments in the resulting ontology, we specified that the test cases should satisfy a couple of requirements:

Compactness. All test cases have to be designed compactly with regards to the amount of OWL constructs they require. On the one hand, this reduces side effects due to language constructs that are not in the focus of the test case. On the other hand, it improves the readability of the test cases in the generated ontology.

Independence. All test cases shall be defined as independently as possible in order to avoid that they interfere with each other. However, this goes along with a larger number of *helper* constructs in the ontology. For instance, properties can only reasonably be tested if classes are added that the properties are linked with. Since adding a pair of classes for each property would result in a large number of additional ontology elements, we used the same class as a domain for

all properties in OntoViBe [8]. OntoBench builds upon this approach by reusing a domain class several times, but creating a new one once a given number of properties has been linked to that class.

Self-Descriptiveness. The elements for all used OWL language constructs have to be named in a way that eases the verification process. Like OntoViBe, OntoBench names elements according to their role in the benchmark ontology. However, where OntoViBe was static and could thus rely on the uniqueness of self-explanatory, yet non-systematically assigned names, OntoBench introduces a uniform naming scheme that ensures uniqueness of names despite the variation in generated ontologies. This is accomplished by prefixing all elements with the name of their corresponding test case. The suffix of the name indicates the role in the test case. For example, the class that serves as the range for the OWL construct `owl:ReflexiveProperty` is named *OwlReflexiveProperty_Range*.

3.2 OWL Profiles and Specific Test Cases

OWL 1 and 2 define multiple profiles of different expressiveness. These profiles restrict the set of eligible language constructs and the way the constructs can be combined. Some ontology tools do not support all elements defined by OWL but are limited by design to one of the less powerful profiles. Accordingly, OntoBench is able to generate ontologies that are conformant with the selected OWL profiles. It provides a preselection of test cases for OWL Lite and DL as defined in the OWL reference [3] as well as OWL 2 EL, RL and QL from the OWL 2 profiles [17]. There is no separate profile for OWL Full, since it does not contain new OWL constructs in comparison to OWL DL.

Since ontologies can make use of multilingualism, for instance in `rdfs:label` annotations, there are also test cases for this aspect.

4 Implementation as a Web Application

OntoBench is implemented as a web application to ease access and reuse [10].[1] The frontend implementation is based on HTML, CSS, and JavaScript in combination with SemanticUI and jQuery. A REST interface is used for communication with the backend, which is implemented as a Java server using the Spring Framework. This server contains the business logic for generating the customized ontologies by means of the OWL API [9]. Additionally, it manages a database of previously generated benchmark ontologies that can be restored via short URIs. These short URIs are provided for easy reference of the generated ontologies, whereas their long URIs are more persistent and transparent, as they include a list of the features contained in the ontology.

[1] OntoBench is released under the MIT license and available on GitHub at https://github.com/VisualDataWeb/OntoBench A public OntoBench service is available at http://ontobench.visualdataweb.org.

As an example, the main part of the ontology generated for the test case of the OWL construct `owl:AllDisjointClasses` is depicted in Listing 1.1. The order and indentation of all statements in the ontology is determined by the Turtle syntax formatter of the OWL API.

Listing 1.1. Main part of the ontology generated for the test case of the OWL construct `owl:AllDisjointClasses` formatted in Turtle syntax.

```
:AllDisjointClasses_Class1 rdf:type owl:Class .
:AllDisjointClasses_Class2 rdf:type owl:Class .
:AllDisjointClasses_Class3 rdf:type owl:Class .

[ rdf:type owl:AllDisjointClasses ;
owl:members ( :AllDisjointClasses_Class1
:AllDisjointClasses_Class2
:AllDisjointClasses_Class3
)
] .
```

4.1 Graphical User Interface

The graphical user interface (GUI) of OntoBench consists of two panels organized in tabs, one allows to configure the benchmark ontology and the other displays the generated output. Figure 1 shows screenshots of parts of the two panels. The configuration panel lists all eligible features grouped into categories, inspired by the grouping of the OWL 2 quick reference guide [2]. The categories are organized into frames in the GUI, while predefined buttons allow to immediately select certain presets, such as all elements of a category or all elements matching a particular OWL profile.

When the user hits the *generate* button or switches to the *generator* tab, the second panel is opened which displays the generated ontology. The ontology is shown on screen and can be downloaded as a file (cp. Fig. 1). It is provided in Turtle syntax by default, but the user can also chose other OWL serializations from a drop-down menu. OntoBench provides all OWL serializations supported by the OWL API (which are Turtle, Manchester, Functional, OWL/XML, and RDF/XML at the moment). The generated output as well as the endings of the ontology URIs change accordingly, so that a particular serialization can be directly accessed from remote via its URI.

OntoBench has been designed for a target group that is at least somehow familiar with OWL and/or wants to use or learn OWL. In some informal user tests, the user interface was praised for its ease of use. The test users liked that they could create OWL ontologies with only a few clicks and found the user interface very self-explanatory.

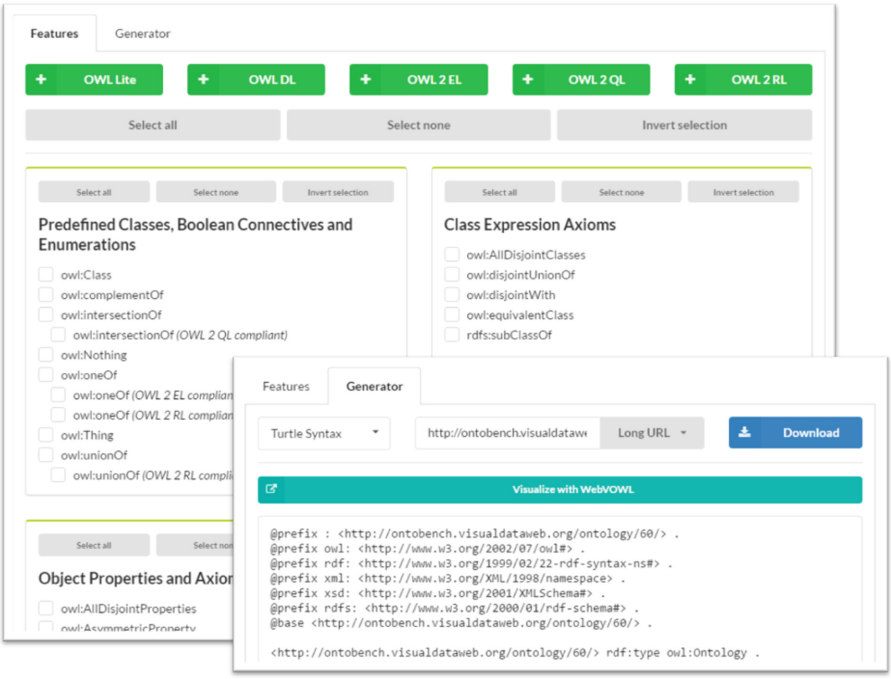

Fig. 1. Screenshots of the user interface of OntoBench showing parts of the two main panels

4.2 Extensibility

In the case that OntoBench does not provide a test case required in a certain situation, users can manually edit and extend the generated ontologies according to their needs. Alternatively, they can edit the source code of OntoBench and add the required test cases to the generator. The source code has an object-oriented design: Each feature is described by a class which is derived from a superclass containing helpers and providing access to the ontology. The feature can either be modeled by directly accessing the OWL API or by using the provided helper classes. Each feature has additionally a name and a token (for the URI) and is assigned to a category for grouping in the user interface. The user interface is automatically generated from the modeled features.

4.3 Limitations

Limitations in using and extending OntoBench result mainly from the OWL API that OntoBench is using to create the OWL ontologies. For instance, the OWL API makes use of the *OWL functional syntax* internally, which represents the OWL constructs `owl:AllDifferent` and `owl:differentFrom` both by the functional concept `DifferentIndividuals`. When having the OWL API output an ontology in Turtle syntax, it will always use `owl:AllDifferent` and

never `owl:differentFrom`, both of which imply the same assertion with two individuals.

5 Validation of the Generated Ontologies

It is not feasible to validate the correctness of the generated ontologies in all possible combinations, but we systematically checked a representative subset using test classes and manual inspection. However, to some extent, we have to trust the OWL API that is used by OntoBench for generating the ontologies. Since it has "widespread usage in a variety of tools" [9] and intends to be a "reference implementation for creating, manipulating and serializing OWL Ontologies"[2], it can be assumed that the generated ontologies are mostly correct in terms of syntax and general structure.

Nevertheless, we applied syntax validators, such as the W3C RDF Validation Service [14], to the representative subset of generated ontologies. The tests showed that all ontologies were valid RDF documents. To validate whether the contents of the generated ontologies are correct, we tested the representative subset by loading the ontologies into different tools, including ontology editors like Protégé and reasoners like Pellet. These checks all showed that the generated ontologies are correct and contain the test cases that were selected in the user interface.

The presets matching OWL profiles were evaluated with the validators built into the OWL API and further refined by manual inspection and comparison with the OWL profiles specifications [3,17]. Some issues with the OWL API were found during these validations and reported to the issue tracker of that project on GitHub.[3] They could be quickly fixed by the developers of the OWL API so that we could finally include the corrected version of the API in OntoBench.

The scalability of OntoBench was tested by selecting different subsets of test cases in the user interface and run the generator. The ontologies are not cached but generated at runtime, which takes less than two seconds on the public OntoBench instance we provide, even if all elements or a large subset of them are selected. The resulting ontology can consist of more than 2000 lines in Turtle syntax in those cases.

6 Application in a Visualization Use Case

During the development of the latest version of the ontology visualization tool WebVOWL [11], we regularly used OntoBench to check whether WebVOWL displays all OWL language constructs according to the VOWL 2 specification [13]. Testing the generated ontologies with WebVOWL was very convenient, as we only had to append their URIs to the URL of WebVOWL. Since we noticed that a visualization like VOWL can help to better understand the generated ontologies, we integrated it into OntoBench by adding a button to the generator panel that directly opens the WebVOWL visualization of each ontology (cf. Fig. 1).

[2] http://owlapi.sourceforge.net.
[3] https://github.com/owlcs/owlapi/issues/435.

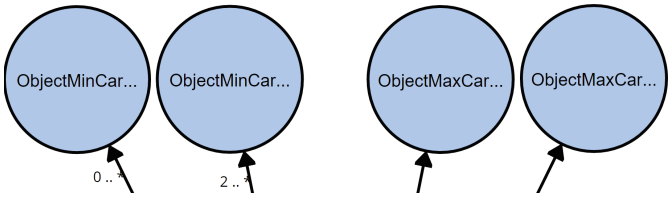

Fig. 2. Example issue found in the use case: while `owl:minCardinality` was shown in the beta version of WebVOWL as expected, `owl:maxCardinality` was not shown at all

In one of the test cases for WebVOWL, we generated an ontology with OntoBench containing all OWL language constructs that are supported according to the VOWL 2 specification [13]. We could uncover two issues this way: (1) The VOWL 2 specification states that `owl:Nothing` should either be visualized the same way as `owl:Thing` or should not be visualized at all, the latter being recommended. However, when we visualized the generated ontology with a beta version of WebVOWL, we discovered that `owl:Nothing` was incorrectly displayed as an external class. (2) While inspecting the indicated cardinalities in the visualization, we realized that in contrast to `owl:minCardinality`, `owl:maxCardinality` was not displayed at all in the beta version of WebVOWL (cf. Fig. 2). As can be seen in the figure, the generated names help to spot the concepts in the visualization and to interpret them correctly.

OntoBench can be extended to also include cases for ABox testing, without affecting the general approach. For instance, we added some specific test cases for the WebVOWL tool, among others a test case generating 1000 instances for a class. However, there is a high variety of such ABox test cases and its systematic investigation constitutes a considerable research effort that would warrant a separate project.

7 Conclusion

The application example illustrates how easy it is to test ontology tools like WebVOWL with OntoBench. Instead of creating suitable test cases from scratch with quite some effort, or searching for existing ontologies that contain those or similar cases, OntoBench allows to quickly generate tailored and consistent ontologies with only a few clicks.

References

1. Ontology alignment evaluation initiative. http://oaei.ontologymatching.org
2. Bao, J., Kendall, E.F., McGuinness, D.L., Patel-Schneider, P.F.: OWL 2 web ontology language quick reference guide (2nd edn.) (2012). https://www.w3.org/TR/owl2-quick-reference/

3. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: OWL web ontology language reference. In: W3C Recommendation (2004). http://www.w3.org/TR/2004/REC-owl-ref-20040210/
4. Bizer, C., Schultz, A.: The Berlin SPARQL benchmark. Int. J. Semant. Web Inf. Syst. **5**(2), 1–24 (2009)
5. Carroll, J.J., Roo, J.D.: OWL web ontology language test cases (2004). http://www.w3.org/TR/owl-test/
6. Guo, Y., Pan, Z., Heflin, J.: LUBM: a benchmark for OWL knowledge base systems. Web Semant. **3**(2–3), 158–182 (2005)
7. Haag, F., Lohmann, S., Negru, S., Ertl, T.: OntoViBe: an ontology visualization benchmark. In: International Workshop on Visualizations and User Interfaces for Knowledge Engineering and Linked Data Analytics (VISUAL 2014), CEUR-WS, vol. 1299, pp. 14–27 (2014)
8. Haag, F., Lohmann, S., Negru, S., Ertl, T.: OntoViBe 2: advancing the ontology visualization benchmark. In: Lambrix, P., et al. (eds.) EKAW 2014. Lecture Notes in Artificial Intelligence (LNAI), vol. 8982, pp. 83–98. Springer, Heidelberg (2015). doi:10.1007/978-3-319-17966-7_9
9. Horridge, M., Bechhofer, S.: The OWL API: a java API for OWL ontologies. Semant. Web **2**(1), 11–21 (2011)
10. Link, V., Lohmann, S., Haag, F.: OntoBench: ontology benchmark generator (2016). http://ontobench.visualdataweb.org
11. Lohmann, S., Link, V., Marbach, E., Negru, S.: WebVOWL: web-based visualization of ontologies. In: Lambrix, P., et al. (eds.) EKAW 2014. Lecture Notes in Artificial Intelligence (LNAI), vol. 8982, pp. 154–158. Springer, Heidelberg (2015). doi:10.1007/978-3-319-17966-7_21
12. Ma, L., Yang, Y., Qiu, Z., Xie, G., Pan, Y., Liu, S.: Towards a complete OWL ontology benchmark. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 125–139. Springer, Heidelberg (2006). doi:10.1007/11762256_12
13. Negru, S., Lohmann, S., Haag, F.: VOWL: visual notation for OWL ontologies (2014). http://purl.org/vowl/spec/v2/
14. Prud'hommeaux, E., Lee, R.: W3C RDF validation service (2004). http://www.w3.org/RDF/Validator
15. Schmidt, M., Hornung, T., Meier, M., Pinkel, C., Lausen, G.: Sp^2bench: a SPARQL performance benchmark. In: Virgilio, R.D., Giunchiglia, F., Tanca, L. (eds.) Semantic Web Information Management, pp. 371–393. Springer, Heidelberg (2009)
16. Smith, M., Horrocks, I., Krötzsch, M., Glimm, B.: OWL 2 web ontology language conformance (2nd edn.) (2012). http://www.w3.org/TR/owl2-conformance/
17. W3C OWL Working Group: OWL 2 web ontology language profiles (2nd edn.) (2012). https://www.w3.org/TR/owl2-profiles/
18. Wang, S.-Y., Guo, Y., Qasem, A., Heflin, J.: Rapid benchmarking for semantic web knowledge base systems. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 758–772. Springer, Heidelberg (2005). doi:10.1007/11574620_54