SHARK: A Test-driven Framework for Design and Evolution of Ontologies

Gustavo Correa Publio[1]

Universität Leipzig, Leipzig, Germany, Institut für Informatik, AKSW Group
gustavo.publio@informatik.uni-leipzig.de

Abstract. In the Semantic Web, the sharing and reuse of knowledge are made possible by ontologies which establish common vocabularies and semantic interpretations of terms. Over the last years, the LOD cloud has been growing substantially in size of each ontology and the total number of objects. In order to ensure a certain level of data quality, several methods have been proposed so far, with different characteristics and approaches, demanding the composition of different tools to ensure a full validation and continuous integration with hosting solutions. In this paper, we present SHARK, a single framework capable to test an ontology using formally pre-defined guidelines or custom SHACL tests, and can also be used for continuous testing during the ontology development process.

Keywords: Ontology evolution, test driven, ontology development, SHACL

1 Introduction

The Linked Open Data Cloud [2] is growing exponentially in the last few years. Besides the increase in the number of available datasets, each of them also evolves and grows in size and amount of data in an accelerated speed, often in a crowd-sourced and collaborative environment – which makes virtually impossible to find an error-free dataset. In order to achieve a sustainable growth, such amount of data demands methods for effective control of data quality and conformance.

Over the last years, many approaches aiming the description and validation of ontologies have been proposed, including methodologies, frameworks, tools, and languages. Among the last group, we can cite the Shape Expression (ShEx) [3], a language that attempted to develop the same function for RDF graphs that languages like XML Schema do for XML; and the Shapes Constraint Language (SHACL) [9], created in 2014 by the W3C Working Group on RDF Data Shapes [1].

In "*Validating RDF Data book*" [7], authors demonstrate several SHACL applications, including "Ontology Validation with SHACL" - a starting point to this work.

Aiming to a new approach for data validation that benefits from SHACL advantages, we present SHARK (SHACL Reasoning over Knowledge-graphs), a

SHACL-based ontology validation framework. This framework aims to 1) provide a continuous-integration test solution for ontology evaluation; 2) support ontology review and design through a simple web interface, by allowing preselected tests based on several ontology development guidelines to be run against user-submitted ontologies; and 3) allow advanced users to run their own custom SHACL tests against any ontology, through the web interface or web service API.

The remainder of this paper is organised as follows. Section 2 reviews the related literature. The research problems and expected contributions are presented in Section 3. Section 4 outlines the research methodology and approach. Section 5 discuss the preliminary tests, while Section 6 discuss the evaluation plan. The paper is finally concluded in Section 7.

2 State of the art

Several languages, methodologies, frameworks, and tools have been proposed in the last twenty years to evaluate ontologies. Tomaszuk [17] makes a comparison of different languages for RDF Validation including SHACL [9], as well as ReSh [16], DSP [4], SPIN [8], OWL [6] and RDFs [5]. In the evaluation, SHACL stands out as the fastest and most complete processing language.

In [14] and as of 2012, the author made an extensive literature review that found relevant dependencies between ontology development methodologies,[1] ontology evaluation frameworks[2] and evaluation tools.[3] The author also showed that although ontology development methodologies had served as key reference for ontology evaluation (mainly by spreading the use of *Competency Questions* - CQ), they were not a requirement for the implementation of evaluation frameworks and tools.

In general lines, CQs consist in questions that a specific ontology must be able to answer. In [15] authors claims that it is difficult to either specify the requirements for an ontology, or to test their satisfaction. Thus, they propose a novel approach to address this problem by leveraging the ideas of competency questions and test-before software development. A key benefit of this approach is to easily guide ontology authoring, especially for authors that are not proficient in logic.

In [14] is also proposed OOPS! (OntOlogy Pitfall Scanner!), a diagnosis tool to both detect potential errors (*pitfalls*) in ontologies and recommend some tips to repair them. The detection methods are mainly based on structural pattern matching and linguistic analysis. OOPS counts with a catalog of errors obtained by manual review where each error is classified as critical, important or minor according its importance level. Although OOPS provides a wider range of known ontology issues, it has some drawbacks: `a)` its catalog of *pitfalls* is not exhaustive,

[1] The seminal work of Gruninger and Fox, Methontology, On-To-Knowledge, DILIGENT and Neon.

[2] OntoClean, OntoQA, Unit Tests, OQuaRE, Neon Guidelines, etc.

[3] ODEClean, ODEval, AEON, Eyeball, Moki, OQuare, OntoCheck, XD-Analyzer.

`b)` it only deals with the conceptual schema level of OWL DL ontologies (TBox) and `c)` it only checks explicit knowledge, i.e. it does not perform inference. In addition, since those *pitfalls* were not been formalized, the tool cannot be integrated smoothly into a *continuous ontology quality assurance process*.

Unit Tests, one of the ontology evaluation frameworks enumerated above, is a well-known technique in test-driven software development and has also been applied to the ontology evaluation field. In a framework initially proposed in [19] and enhanced in [18], the working ontology is validated against a positive and a negative test ontology. Each unit test controls that each axiom in the positive test ontology be inferred by the working ontology and that each axiom in the negative test ontology be not inferred by the working ontology. Thus, if a unit test fails, an error might exist in the working ontology. Notice that the proposed framework formalizes competency questions but does not formalize all ontology requirements.

In [10], a test-driven approach to evaluate Linked Data quality is described. This approach encodes data quality constraints in generic SPARQL query templates which in turn are instantiated automatically into specific quality test queries for a particular ontology or dataset. The data quality constraints are compiled as a reusable set of Data Quality Test Patterns (DQTP) ready to be integrated as automated test-methods.

3 Problem statement and contributions

The following examples illustrates some of the problems that might raise during the ontology development process. As stated in [15], authoring ontologies is a non-trivial task as ontology authors are usually domain experts but not necessarily proficient in logic. This may cause misleading in following the best practices in ontology authoring, as for instance missing useful metadata. In fact, [11] shows that a significant part of datasets available in the LOD cloud lack or has incorrect/incomplete basic metadata information. Besides the metadata at the dataset level, according to [13] the lack of label or comment in the schema level (classes and/or properties) also deviates best practices.

In [12], authors define two distinct modes of ontology evolution: *traced* and *untraced* evolution. In the first mode we treat the evolution as a series of documented changes in the ontology. Those can be structural changes in the schema level (TBox) or in the individuals of the ontology (ABox). On the other hand, due to the extremely distributed nature of ontologies, we must also account for the fact that we will not always have the trace of changes that led from one version to another, leading to the untraced mode. In any case, it is expected that the data keeps consistent in the new version of the ontology in order to keep the compatibility between those versions, but such consistency is difficult to be assured.

Finally, impacts of ontology evolution may be of interest to data authors, consumers, reviewers, system developers, and so on, but they might be difficult to trace, especially in crowd-sourced ontology edition. An approach to preview or

detect such impacts in any ontology must be public available to general purpose, so users can try to mitigate such impacts prior to publish a new version of the ontology.

3.1 Research questions

To solve the identified problems in the ontology design and evolution process, the following research questions are intended to be addressed with the proposed approach:

1. Regardless of prior technical knowledge in logic, authoring or programming languages, how can any user evaluate whether his or her ontology addresses the best practices of ontology design w.r.t. the data structure, metadata, and so on?
2. How can non-conformance data inserted between two distinct versions of ontology (whether in traced or untraced evolution) be detected at ABox or TBox level?
3. In order to assure data quality, how can ontology evolution be evaluated prior to final publication?
4. In a community-based crowd-sourced environment, how is it possible to mitigate data quality issues?

3.2 Contributions

In order to solve the mentioned research questions, this paper presents the SHARK framework with the following respective contributions:

1. a novel web-based application that a) allows users to select relevant tests among predefined guidelines based on best practices of ontology design, and b) allows users to run custom SHACL tests against any given ontology;
2. for any given ontology and test set, a report at instance level for every violation found;
3. a web API enabled through a web service endpoint that allows users to run tests against any ontology at any phase of development;
4. a continuous-integration test solution for ontologies hosted in version-control environments (e.g. GitHub).

4 Research Methodology and Approach

To achieve the intended contribution goals, the SHARK framework consists in an environment as pictured in Figure 1. The only external dependency, that was not developed in the context of this work, is the RDFUnit tool[4], which provides the SHACL implementation, running the SHACL tests in the provided ontology and giving the results.

[4] `http://rdfunit.aksw.org/`

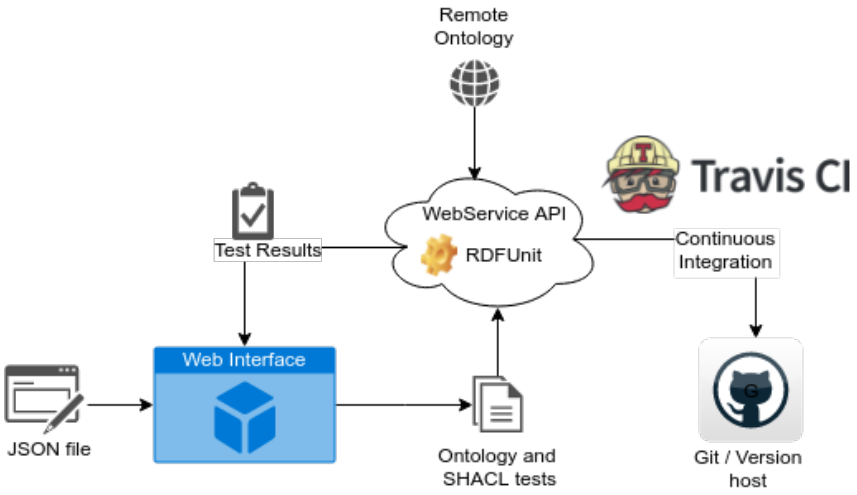

Fig. 1. The SHARK Framework architecture

4.1 Technologies

The front-end Web application under development uses HTML5 and Javascript ES6, and also the Bootstrap framework.[5] The Web Service and its API is being written in Java, with support of Springboot framework.[6] It runs in the backend the RDFUnit tool. Finally, the developed continuous integration system integration was preliminary tested with Travis-CI[7] (as shown in Figure 3) and GitHub[8] for hosting and version control.

4.2 Web application home

The web application consists of a three-step-process. As seen in Figure 2 `[a]`, the initial screen of the web-based application offers to the user to select between uploading an ontology from his local computer or specifying the remote URL of the ontology to be tested.

In the second step (Figure 2 `[b]`), users can either select which pre-defined guidelines they want to test against the provided ontology or define custom SHACL tests in the second tab (Figure 2 `[c]`). There are 20 distinct pre-defined guidelines that can be tested in the provided ontology. To make it easier to update such guidelines, they are described in a separated JSON file that contains the SHACL tests and generates the HTML form. When selecting each of them, the user is indirectly generating (internally by the tool) a SHACL test which

[5] `https://getbootstrap.com/`
[6] `https://spring.io/`
[7] `https://travis-ci.org`
[8] `https://github.com/`

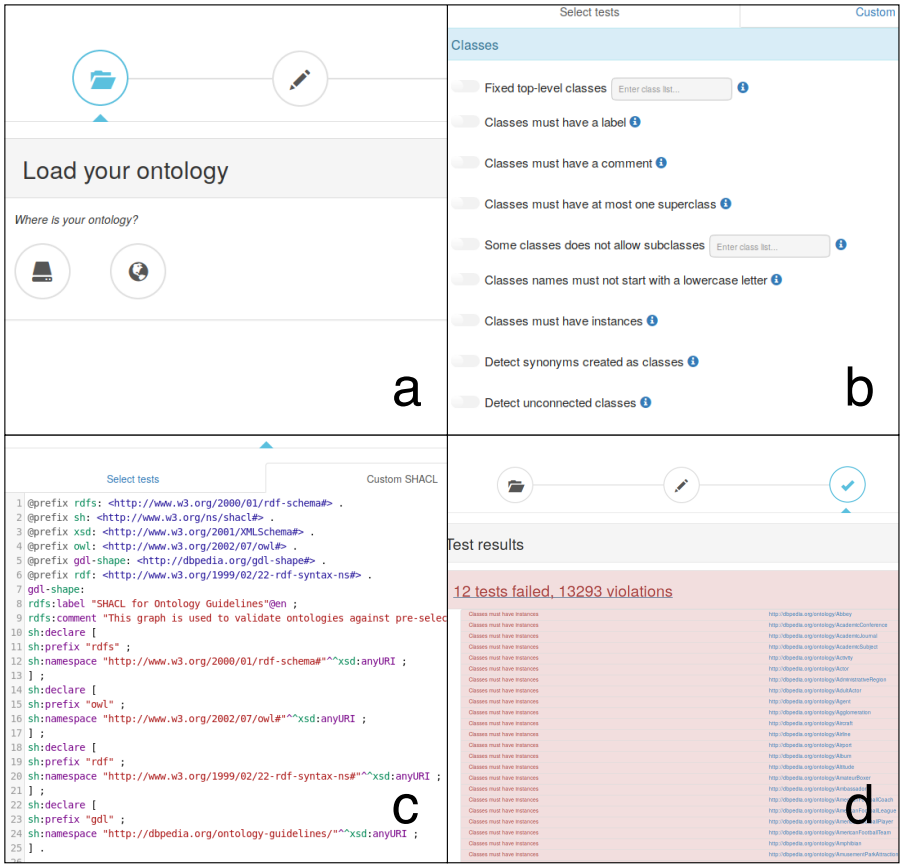

Fig. 2. The SHARK web tool interface, with a) ontology selection screen, b) guidelines selection, c) custom SHACL editor and d) test results report.

can be either a SHACL Core-based test or a SHACL SPARQL-based one. Such tests were designed based on the best practices reviewed in the literature, such as OOPS! framework[14], or those included in *Validating RDF Data book* [7].

To the users, there are two different types of tests: those that they can only select and activate, and those that demands an extra, custom text input. The latter are the "*Fixed top-level classes*" test, where users can specify whether the first layer of the class hierarchy of the ontology is fixed and cannot be changed by simply listing those fixed classes, and the "*Some classes does not allow subclasses*" test, which in the same way, requires as text input a list of classes that does not allow subclasses.

Finally, in the third step (Figure 2 `[d]`), the report with the test results is presented, showing all the elements that violates any of the selected guidelines/SHACL tests.

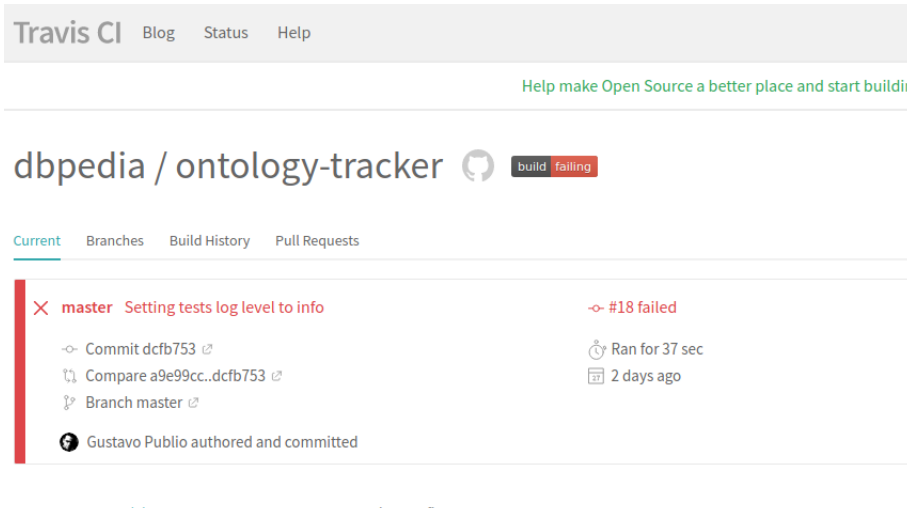

Fig. 3. The SHARK continuous integration, showing a build that failed due to a test violation.

4.3 RESTful API

The RESTful Web Service API is described in the SwaggerHub service.[9] It provide methods that supports different input formats of the ontology file, as well as all output formats that Apache Jena[10] supports.

5 Preliminary Results

5.1 Guidelines evaluation

In order to evaluate the usefulness of the pre-selected guideline tests, we run most of them (18 out of 20 - except the two that requires custom user input) against the ontologies available in the LOV Cloud.[11] As provided by LOV vocabulary list API,[12] we found 633 ontologies in the cloud. From those, 237 ontologies were not tested due to different reasons: broken (Error 404) or non-public (Error 401) URI, connection timeout, empty ontologies (model with 0 elements), etc. Thus, 396 ontologies were effectively tested.

Given that the tested ontologies are public available in the LOV list, and that this test set consists in ontologies of different domains and authors, the idea is that each of the designed tests should neither fail or pass in 100% of the ontologies, otherwise that test would not be useful for generic guidelines purposes.

[9] `https://app.swaggerhub.com/apis/gcpdev/SHARK/0.1`
[10] `http://jena.apache.org/documentation/io/rdf-output.html#normal-printing`
[11] `https://lov.okfn.org`
[12] `http://lov.okfn.org/dataset/lov/api/v2/vocabulary/list`

Table 1 shows the test results. The most frequent violation is classes/properties must have a label/comment, which shows that unfortunately the metadata of ontology elements still lack in most of the available datasets. On the other hand, tests like "*Detect relationships inverse to themselves*", "*Detect cycles in the class hierarchy*" or "*Detect synonyms created as classes*" have only a few occurrences, but they point out critical structural problems in each ontology they occur.

6 Evaluation Plan

6.1 User Interface evaluation

In order to obtain a deeper, critical evaluation of the SHARK web based tool, we plan to make an evaluation of its web user interface with a group of users through surveys, rating the usability and relevance/usefulness of those predefined guidelines.

6.2 Improve guidelines

The literature review and guidelines maintenance will be periodically revisited, in order to improve the actual tests according to users feedback and add new predefined guidelines.

6.3 Better evolution/alignment evaluation

The evolution process of ontologies is very related to the alignment process, as both of them deals with the differences and compatibility between two different ontology versions. Both processes demand a better evaluation, with study cases and practical examples of the impact of the SHACL tests over each of them.

7 Conclusion

In this paper, we presented SHARK, a test-driven framework for design and evolution of ontologies. The framework consists in a Web-based interface where users can either choose between 20 different pre-defined guidelines to test their ontology, or run their own custom SHACL tests. The pre-defined tests were run successfully against 396 different ontologies, demonstrating their usefulness against real data. The framework has also an web service API, and a continuous integration system setup for automatic evaluation of ontology evolution.

Finally, all the SHARK framework code is open-source and available in the GitHub.[13]

[13] `https://github.com/gcpdev/shark`

Table 1. Compilation of test results over 396 different ontologies streamed from the LOV Cloud. The second column shows the absolute number of ontologies where the test failed; the third one shows the average frequency of element violations, and the last column shows the percentage of ontologies that failed the test with at least one violation

Test	#Occurr.	Av. Freq. (%)	Ontologies occurr. (%)
Classes must have a label	363	12.04	92.37
Classes must have a comment	363	12.04	92.37
Classes must have at most one superclass	170	3.27	43.26
Classes names must not start with a lowercase letter	4	3.87	1.02
Classes must have instances	282	9.46	71.76
Detect synonyms created as classes	12	0.75	3.05
Detect unconnected classes	129	1.14	32.82
Detect cycles in the class hierarchy	14	0.91	3.56
Properties must have a label	359	15.26	91.35
Properties must have a comment	359	15.26	91.35
Properties must have at most one domain	68	0.93	17.30
Properties must have at least one class as domain	359	20.77	91.35
Properties must have at most one range	62	0.65	15.78
Properties must have at least one class as range	359	20.77	91.35
Properties must have at most one superproperty	60	2.83	15.27
Properties names must not start with a capital letter	19	4.23	4.83
Detect relationships inverse to themselves	15	0.50	3.82
Detect wrongly defined relationship 'is'	23	0.36	5.85

Acknowledgements

This paper's research activities were funded by grants from the Smart Data Web BMWi project (GA-01MD15010B) and CNPq foundation (scholarship 201808/2015-3). The author also acknowledges the support of Adam Sanchez, Sebastian Hellmann, Magnus Knuth, and Rafael Peñaloza in the development of this paper.

References

1. Rdf data shapes working group, https://www.w3.org/2014/data-shapes/wiki/Main_Page, last accessed 29th January 2018
2. Abele, A., McCrae, J.P., Buitelaar, P., Jentzsch, A., Cyganiak, R.: Linking open data cloud diagram 2017 http://lod-cloud.net/
3. Boneva, I., Gayo, J.E.L., Hym, S., Prud'hommeau, E.G., Solbrig, H., Staworko, S.: Validating rdf with shape expressions. arXiv preprint arXiv:1404.1270 (2014)
4. Bosch, T., Eckert, K.: Towards description set profiles for rdf using sparql as intermediate language. In: International Conference on Dublin Core and Metadata Applications. pp. 129–137 (2014)
5. Brickley, D., Guha, R.V.: Resource description framework (rdf) schema specification 1.0: W3c candidate recommendation 27 march 2000 (2000)
6. Dean, M., Schreiber, G., Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: Owl web ontology language reference. W3C Recommendation February 10 (2004)
7. Gayo, J.E.L., Prud'Hommeaux, E., Boneva, I., Kontokostas, D.: Validating RDF Data, vol. 7. Morgan & Claypool Publishers (2017)
8. Knublauch, H., Hendler, J.A., Idehen, K.: Spin-overview and motivation. W3C Member Submission 22 (2011)
9. Knublauch, H., Kontokostas, D.: Shapes constraint language (shacl), https://www.w3.org/TR/shacl/, last accessed 29th January 2018
10. Kontokostas, D., Westphal, P., Auer, S., Hellmann, S., Lehmann, J., Cornelissen, R., Zaveri, A.: Test-driven evaluation of linked data quality. In: 23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014. pp. 747–758 (2014), http://doi.acm.org/10.1145/2566486.2568002
11. Neto, C.B., Kontokostas, D., Kirschenbaum, A., Publio, G.C., Esteves, D., Hellmann, S.: Idol: Comprehensive & complete lod insights. In: Proceedings of the 13th International Conference on Semantic Systems. pp. 49–56. Semantics2017, ACM, New York, NY, USA (2017), http://doi.acm.org/10.1145/3132218.3132238
12. Noy, N.F., Klein, M.: Ontology evolution: Not the same as schema evolution. Knowledge and information systems 6(4), 428–440 (2004)
13. Poveda, M., Suárez-Figueroa, M.C., Gómez-Pérez, A.: Common pitfalls in ontology development. In: Conference of the Spanish Association for Artificial Intelligence. pp. 91–100. Springer (2009)
14. Poveda-Villalón, M.: Ontology Evaluation: a pitfall-based approach to ontology diagnosis. Ph.D. thesis, Departamento de Inteligencia Artificial, Universidad Politécnica de Madrid (2 2016)
15. Ren, Y., Parvizi, A., Mellish, C., Pan, J.Z., Van Deemter, K., Stevens, R.: Towards competency question-driven ontology authoring. In: European Semantic Web Conference. pp. 752–767. Springer (2014)
16. Ryman, A.G., Le Hors, A., Speicher, S.: Oslc resource shape: A language for defining constraints on linked data. LDOW 996 (2013)
17. Tomaszuk, D.: Rdf validation: A brief survey. In: International Conference: Beyond Databases, Architectures and Structures. pp. 344–355. Springer (2017)
18. Vrandecic, D.: Ontology Evaluation. Ph.D. thesis, Departamento de Inteligencia Artificial, Fakultat fur Wirtschaftswissenschaften des Karlsruher Instituts fur Technologie (KIT) (6 2010)
19. Vrandecic, D., Gangemi, A.: Unit tests for ontologies. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM Workshops (2). Lecture Notes in Computer Science, vol. 4278, pp. 1012–1020. Springer (2006)