Ontolo-CI: Continuous Data Validation With ShEx

Gustavo Correa Publio[1,*], Jose Emilio Labra Gayo[2], Guillermo Facundo Colunga[2] and Pablo Menendéz[2]

[1]AKSW Research Group, University of Leipzig, Germany
[2]WESO Research Group, University of Oviedo, Spain

Abstract

The amount of public linked data published on the Web has been growing more and more over the last years. In order to keep the consistency of this continuously-growing base of datasets, data validation is a necessity for data publishers and maintainers. To address such validation of ontologies, there are mainly two shapes-based languages, e.g. ShEx and SHACL. The former is pointed out as a concise, formal, modeling approach, while the second is a W3C recommendation for data validation. SHACL already has available tools to perform validation on the fly, but ShEx still lacks this feature. In order to reduce this gap, this work presents Ontolo-CI: a tool for automated data validation, capable of accepting ShEx shapes as input, allowing users to validate their data on the fly through an CI/CD approach, by using GitHub Actions.

Keywords

shex, data validation, ontology validation, continuous integration

1. Introduction

The RDF data model is a core technology of the Semantic Web. RDF is used to integrate data from heterogeneous sources, is extensible, flexible and can be manipulated with the SPARQL query language [1].

The need to describe the topologies (or shapes) of RDF graphs triggered the creation of an early version of Shape Expressions (ShEx) and the formation of a World Wide Web Consortium (W3C) Working Group—the Data Shapes Working Group—in 2014 [2]. Its task was to recommend a technology for describing and expressing structural constraints on RDF graphs. This has led to SHACL [3] and further development of ShEx. But defining, developing, and extending ontologies, as stated in [4], is a non-trivial task as ontology authors are usually domain experts but not necessarily proficient in logic. The usage of collaboration tools, such as versioning systems, may address part of the issues, but a continuous validation process is needed to validate data on the fly, i.e., right after each change, to assure that data is still consistent with its schema after each iteration.

SEMANTICS 2022 EU: 18th International Conference on Semantic Systems, September 13-15, 2022, Vienna, Austria
*Corresponding author.
gustavo.publio@informatik.uni-leipzig.de (G. C. Publio); labra@uniovi.es (J. E. L. Gayo); thewilly.work@gmail.com (G. F. Colunga); pabloyo97@hotmail.com (P. Menendéz)
https://aksw.org/GustavoPublio (G. C. Publio); https://labra.weso.es/ (J. E. L. Gayo)
0000-0002-3853-3588 (G. C. Publio); 0000-0001-8907-5348 (J. E. L. Gayo); 0000-0003-1283-2763 (G. F. Colunga); 0000-0002-8602-6927 (P. Menendéz)

For this purpose, SHACL has SHARK [5], a SHACL-based ontology validation framework. But ShEx still misses a continuous-integration and continuous-development (CI/CD) approach for validation of ontologies on the fly. To address this gap, in this paper we will present Ontolo-CI, a CI/CD tool based in GitHub Actions which is capable of running tests and validating RDF data with ShEx automatically for a given GitHub repository.

The paper is divided as following: Section 2 describes some related work. Section 3 describes the features, architecture and general implementation of the Ontolo-CI tool, and finally on Section 4 we present our preliminary conclusion and possibilities of future work.

2. Related Work

Recent works have been trying to tackle the challenge of data validation in collaborative environments with different approaches and technologies. However, to our knowledge, none of them are able to provide ShEx validation with CI/CD capabilities yet.

With focus on biological ontology development (OBO ontology), the ROBOT [6] is a command line tool which can be integrated in custom CI/CD environments and has the option to run validations at logical level (i.e., look for *incoherency*). It also runs validations based in a SPARQL query, but misses the capabilities of a validation language such as ShEx or SHACL.

The OnToology work [7] lies in a similar place: it is capable of being integrated in GitHub repositories (although technically not directly in CI/CD environments), and triggers, besides other features, the evaluation of the new ontology according to OOPS! pitfalls [8], but it lacks the possibility to use a validation language with custom test cases.

Another approach, the eXtreme Design methodology (XD), has TESTaLOD [9], a tool designed for supporting the testing team of XD projects - but although it is able to read files from a Git repository, it cannot be automatically integrated to Git environments, and is only able to validate tests written with the TestCase OWL meta mode [10].

Finally, SHARK [5] is the closest to what we have proposed. It uses the Travis-CI to run pre-defined or custom SHACL tests over an uploaded ontology file, and publishes the ontology in a GitHub repository. As a WebService, it is able to be integrated to a CI/CD environment such as GitHub actions, although prior configuration would be necessary.

3. Implementation

Ontolo-CI[1] is a docker based system that integrates with GitHub to provide a continuous integration system for ontologies. In order to achieve that, it uses Shape Expressions and test instances for data validation.

The tool is focused on develop continuous integration for ontologies, although inspired in Travis-CI and many other continuous integration systems. For that, it enables authors that uses GitHub as version control system to add it as a check to Pull Requests or Pushes to different branches via GitHub Actions.

In fig. 1, it is demonstrated an abstraction of Ontolo-CI. As can be seen, it can be deployed as a docker container in any machine. Then it will listen to GitHub webhooks. Whenever a

[1]Available in https://github.com/weso/ontolo-ci

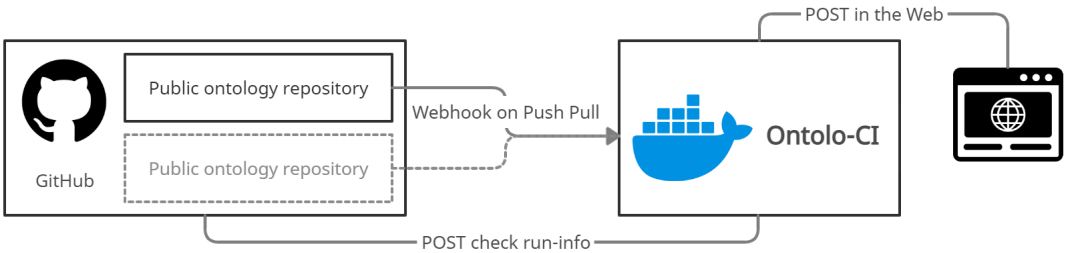

Figure 1: Overview of the Ontolo-CI architecture.

webhook from GitHub arrives, it immediately schedules a build. After the build is finished, Ontolo-CI will notify GitHub and publish the data on its web page.

3.1. Architecture & Technical details

Its implementation is mainly in Java, with the support of Docker[2] containers in order to achieve easiness of execution regardless the user's platform. For the web module, the JavaScript language with the support of React.js[3] component were used. Finally, a NoSQL database instance of MongoDB[4] was also used to store the results of each run, as can be seen in fig. 2.

In order to achieve scalability, each module work independently, in a microservices architecture, while the whole builds up the system's functionalities. Those modules are:

- ***Listener***: The listener component receives notifications from GitHub when a Pull Request is started or when commits are pushed. This notifies the scheduler about the new build to perform.
- ***Hub***: It acts as a GitHub API interface client. It allows the system to collect files from GitHub but also to inform about the status of the builds.
- ***Scheduler***: This component receives builds to schedule from the listener, then creates a worker with the build and schedules its execution.
- ***Worker***: Each worker contains a build to execute. A build is a set of tests to execute over an ontology. It only knows how to execute tests when told and who to notify when finished.
- ***Database***: When a build is finished by a worker, the results of the build are stored in a database. Up to now the results that are being stored are: repository, branch, event, result. The result stores not only the results of the test cases, but also the execution time and other metrics.
- ***API***: The API provides an access layer for third party services that need to explore the data from an Ontolo-CI instance. It is also used by the web service. It only allows reading data at the time.

[2]https://www.docker.com/
[3]https://reactjs.org/
[4]https://www.mongodb.com/

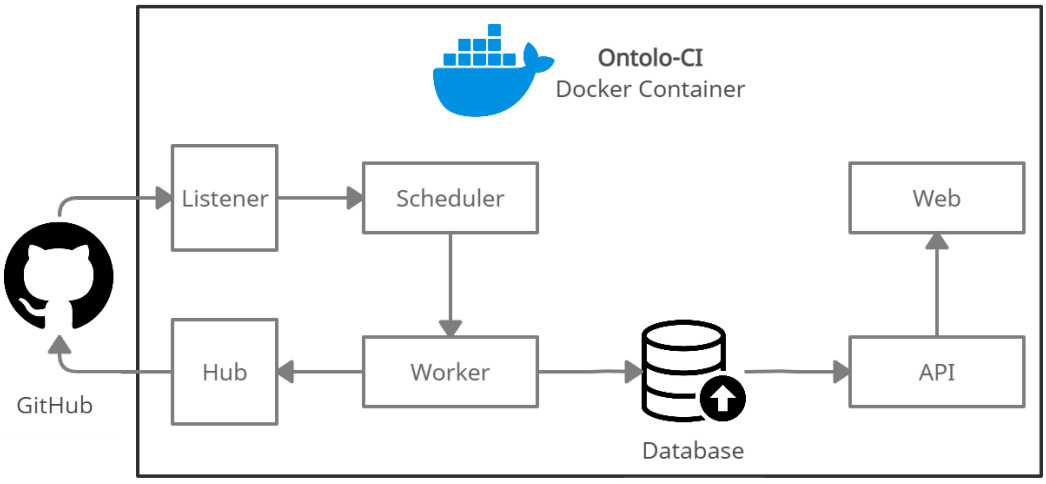

Figure 2: A closer look at the Ontolo-CI architecture.

- ***Web*** Provides the interface that displays to the user the results of all scheduled executions.

3.2. Features

For every run, there are a couple of features to provide the results to the user, as listed below:

- ***GitHub Check Runs***: Ontolo-CI uses GitHub Check Runs[5] in order to provide detailed feedback on commits. Every time a push or a pull request is made, Ontolo-CI creates and updates a check run with the status of the validation process.
- ***GitHub Details View***: Once the validation has finished, a more detailed view can be seen in the details section of the check run.
- ***Website Dashboard view***: The Dashboard View shows in the website page all the builds that has been executed over the Ontolo-CI instance. It includes information such as the name of the repository where the build comes from, Owner of the repository, commit message and ID, branch name and number, and date and time of the execution.
- ***Website Build-specific view***: The user can get details of the run in a build-specific view for each build. This view shows all the test cases that make up the build. The users can see the details of the validation process for any test. The detailed view of a test case shows the validation and the expected result for each node with its shape. It is also possible to see the full shape map result of the test case.

More details and illustrations on the above features can be found in the Ontolo-CI website.[6]

[5] https://docs.github.com/en/rest/checks/runs
[6] https://www.weso.es/ontolo-ci/

4. Conclusion and future work

In this work, we introduced Ontolo-CI, a tool capable of validating ontologies through ShEx on the fly within a GitHub environment. By using the GitHub Actions, the tool is capable of producing checks and reports for every change in the repository, making sure that new changes does not introduce inconsistencies in the data according to the defined ShEx shapes tests. We plan to extend the tool to enable the validation of SHACL shapes, as well as its features, transforming it into a more comprehensive Ontology Validation Framework.

Acknowledgments

Gustavo Correa Publio acknowledges the Schwarz IT KG for the sponsorship of his participation in the conference.

References

[1] J. E. L. Gayo, E. Prud'Hommeaux, I. Boneva, D. Kontokostas, Validating rdf data, Synthesis Lectures on Semantic Web: Theory and Technology 7 (2017) 1–328.

[2] E. Prud'hommeaux, J. E. Labra Gayo, H. Solbrig, Shape expressions: an rdf validation and transformation language, in: Proceedings of the 10th International Conference on Semantic Systems, 2014, pp. 32–40.

[3] H. Knublauch, D. Kontokostas, Shapes Constraint Language (SHACL), Recommendation, W3C, 2017. URL: https://www.w3.org/TR/2017/REC-shacl-20170720/.

[4] Y. Ren, A. Parvizi, C. Mellish, J. Z. Pan, K. v. Deemter, R. Stevens, Towards competency question-driven ontology authoring, in: European Semantic Web Conference, Springer, 2014, pp. 752–767.

[5] G. C. Publio, Shark: A test-driven framework for design and evolution of ontologies, in: European Semantic Web Conference, Springer, 2018, pp. 314–324.

[6] R. C. Jackson, J. P. Balhoff, E. Douglass, N. L. Harris, C. J. Mungall, J. A. Overton, Robot: a tool for automating ontology workflows, BMC bioinformatics 20 (2019) 1–10.

[7] A. Alobaid, D. Garijo, M. Poveda-Villalón, I. Santana-Perez, A. Fernández-Izquierdo, O. Corcho, Automating ontology engineering support activities with ontoology, Journal of Web Semantics 57 (2019) 100472.

[8] M. Poveda-Villalón, A. Gómez-Pérez, M. C. Suárez-Figueroa, Oops!(ontology pitfall scanner!): An on-line tool for ontology evaluation, International Journal on Semantic Web and Information Systems (IJSWIS) 10 (2014) 7–34.

[9] V. A. Carriero, F. Mariani, A. G. Nuzzolese, V. Pasqual, V. Presutti, Agile knowledge graph testing with testalod., in: ISWC (Satellites), 2019, pp. 221–224.

[10] E. Blomqvist, K. Hammar, V. Presutti, Engineering ontologies with patterns-the extreme design methodology., Ontology Engineering with Ontology Design Patterns (2016) 23–50.