TurtleEditor: An ontology-aware web-editor for collaborative ontology development

Niklas Petersen*, Gökhan Coskun*, Christoph Lange*
*University of Bonn
Bonn, Germany
{petersen, coskun, langec}@cs.uni-bonn.de

***Abstract*—Inspired by the shift of vocabulary development projects towards repository hosting services such as GitHub, we noticed the lack of ontology-aware editors that can be easily connected to these repositories. This motivated us to build a web client optimized for the communication with external repositories and including specific functionalities to ease the participation in collaborative ontology development efforts also for non-expert contributors. This paper describes TurtleEditor, an open-source web editor, which can load files from, and commit changes to a central repository and offers features such as syntax highlighting, syntax checking, auto-completion and a SPARQL endpoint to query the ontology.**

I. Introduction

Ontologies defined as a specification of a shared conceptualization reflect the consensus among experts in a certain application domain. They are thus developed in collaborative efforts of domain experts and knowledge engineers. Particularly, the presence of domain experts with little technical background requires tools that ease contribution and give feedback in case of faulty changes. Thus, basis features a tool needs to support include a user-friendly editing surface and the option to query the vocabulary (for testing).

In recent years, more and more large ontology development projects have moved towards repository hosting platforms such as GitHub. (This is especially true for light-weight ontologies with low expressivity, which we call vocabularies.) Such hosting services provide, besides a version control system with the possibility to revert to any previous revision, also project management features for issue tracking, discussion of changes and branching for parallel development. Examples of projects that have moved include *schema.org*[1], *FOAF*[2], *BIBO*[3], *DOAP*[4] and the *Music Ontology*[5].

While recent approaches such as WebProtégé [1] or VocBench [2] have added support for revision histories and the possibility to contribute through a web client, their revision control features are limited. Only VocBench offers a query service (WebProtégé plans to add it [3]) but is optimized for vocabulary projects in the SKOS language.

The article is structured as follows: We outline general requirements as well as requirements of specific use cases in multiple domains in section II. Section III describes in detail our solution, called TurtleEditor, with all its used libraries and protocols. In section IV, we compare TurtleEditor to the existing solutions WebProtégé and VocBench, and finally conclude in section V together with an outlook to future work.

II. Requirements

The development of the TurtleEditor was originally triggered by the requirements of the *MobiVoc consortium*[6]. These requirements were further refined by taking into account the needs of additional vocabulary development projects including SCORVoc for supply chain management, Shopfloor for industrial production lines, and Odette for master data management[7].

The aim of the MobiVoc consortium is to support the mobility of people through the mobility of data by developing a comprehensive vocabulary for all aspects of mobility ranging from map data, over points-of-interest to gas stations, electric charging points and traffic management. The consortium comprises a number of stakeholders (car manufacturers, researchers, IT companies, public administrations), which send representatives with different backgrounds to the working groups.

While teaching the Turtle RDF syntax to domain experts proved feasible in our setting, avoidable faulty commits led us to identify the following requirements for a tool aiming on improving the quality of these contributions:

Syntax Checking: In particular, the environment should tolerate syntax errors, but it should detect them and highlight or report them in a comprehensible way.

Auto Completion: Auto completion for terms of established ontology language such as RDF or RDFS should be made available.

Syntax Highlighting: Each syntactic construct should be highlighted with different colors to increase the readability and improve the comprehensibility of the content.

Load/Commit from/to Repository: Like the GitHub web editor used previously by our domain experts, the new system should also allow loading files from the central repository, and committing and pushing changes.

[1] https://github.com/schemaorg/schemaorg
[2] https://github.com/foaf/foaf
[3] https://github.com/structureddynamics/Bibliographic-Ontology-BIBO
[4] https://github.com/edumbill/doap
[5] https://github.com/motools/musicontology

[6] http://www.mobivoc.org/
[7] See https://github.com/vocol for an overview.

Query Service: To allow executing test queries and to demonstrate the functionality of the vocabulary, a query interface is required.

III. Implementation

A. Key principles

A key principle of our implementation of TurtleEditor is to reuse existing mature solutions that are broadly accepted. Thanks to more and more repository hosting services providing RESTful interfaces, it is feasible to build customized web frontends for their functionality, as explained in section III-B. Similarly, due to the increasing number of available mature JavaScript libraries, our client-side development can focus on the actual Turtle editing, as explained in section III-C.

TurtleEditor is designed to be compatible with existing repository services such as GitHub. Alternative solutions (e.g. BitBucket[8] or GitLab[9]) also provide a REST API and the same features identified as useful for vocabulary development. While we aim at supporting more platforms, as a first step, we focused on GitHub.

It aims at providing an ontology-aware frontend on the top of repository services without forcing experienced users to use it and thus creating a lock-in. One can always clone the vocabulary's repository and work locally with one's preferred ontology editor, or even resort to using GitHub's basic web editor.

Furtermore, as syntax checking is a major requirement for our solution: While the goal is to improve the quality of contributions, we nevertheless still allow commits with errors in them. Influenced by the *wiki way* [4] prinicple, we prefer faulty contributions being added and fixed by others over otherwise useful contributions not being made.

B. Architecture

Figure 1 displays the overall architecture of our proposed solution including the services that a repository hosting service is expected to provide, and for which TurtleEditor offers a frontend. By using the host's REST interface, TurtleEditor is able to load all vocabulary sources in the repository and commit changes. Services such as issue tracking, access control, a wiki for documentation, and the version-controlled repository itself are thus given by the hosting platform. Therefore, TurtleEditor only needs to implement the missing features (See section II) we identified as requirements for our tool.

C. Implementation details

We implemented TurtleEditor by making use of multiple mature open source libraries.

code editor For the basic in-browser text editor, we used the JavaScript library *CodeMirror*[10] as a basis. Started in 2007, CodeMirror focuses on extensibility and offers a rich programming API. Besides being able to handle large texts, it is able to fulfill two of our requirements: syntax highlighting and auto completion. Syntax highlighting is supported natively for more than 100 programming languages; Turtle is one of them. Figure 2 presents on the right side the editor with a live example of a Turtle vocabulary source.

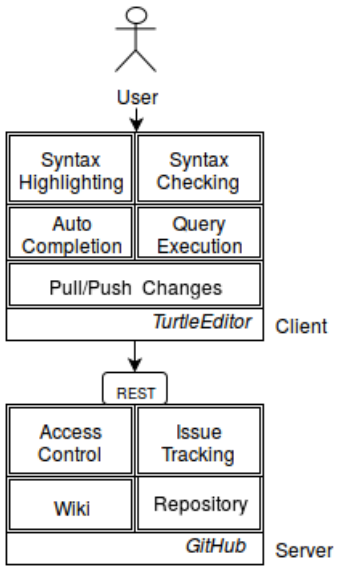

Fig. 1. TurtleEditor Architecture

auto completion CodeMirror allows auto completion by defining the namespaces internally as *hints*. Thus, we defined the `rdfs` terms which are checked once the auto completition is triggered. As an example, we defined the keyboard shortcut `Ctrl+Space` as the trigger event. Once the prefix `rdfs:` is typed and this event is set in motion, TurtleEditor checks internally for the `rdfs` namespace for provides the user with possible terms.

validation Another major requirement is the validation of the vocabulary. The *N3.js*[11] JavaScript library supports manipulation of RDF on the client side. We implemented the parsing feature as follows: Every time CodeMirror detects a change by the author, the entire text source is parsed automatically by N3.js for syntax errors. In case an error is found, the faulty code line is highlighted with a red background, and a red dot is added next to the line number. The author can hover the mouse over the dot and gets an error description in form of a tool tip. This gives the author the chance to fix faulty lines and also to gain a better understanding of the Turtle language. Nevertheless, as mentioned in our key principles, syntax checking is only a soft measure and does not prevent users of committing these changes to the repository.

query service To support querying the live vocabulary version, we used the `rdfstore-js` library[12]. rdfstore-js supports currently SPARQL 1.0 and most of SPARQL 1.1/Update. SPARQL 1.1 query is currently being added to the library. To clearly separate the UI between editing and querying the vocabulary, we added a separate window[13] which displays the

[8] https://bitbucket.org/
[9] https://gitlab.com/
[10] https://codemirror.net/
[11] https://github.com/RubenVerborgh/N3.js
[12] https://github.com/antoniogarrote/rdfstore-js
[13] https://rawgit.com/vocol/vocol/master/SparqlProcessor/sparql-processor.html

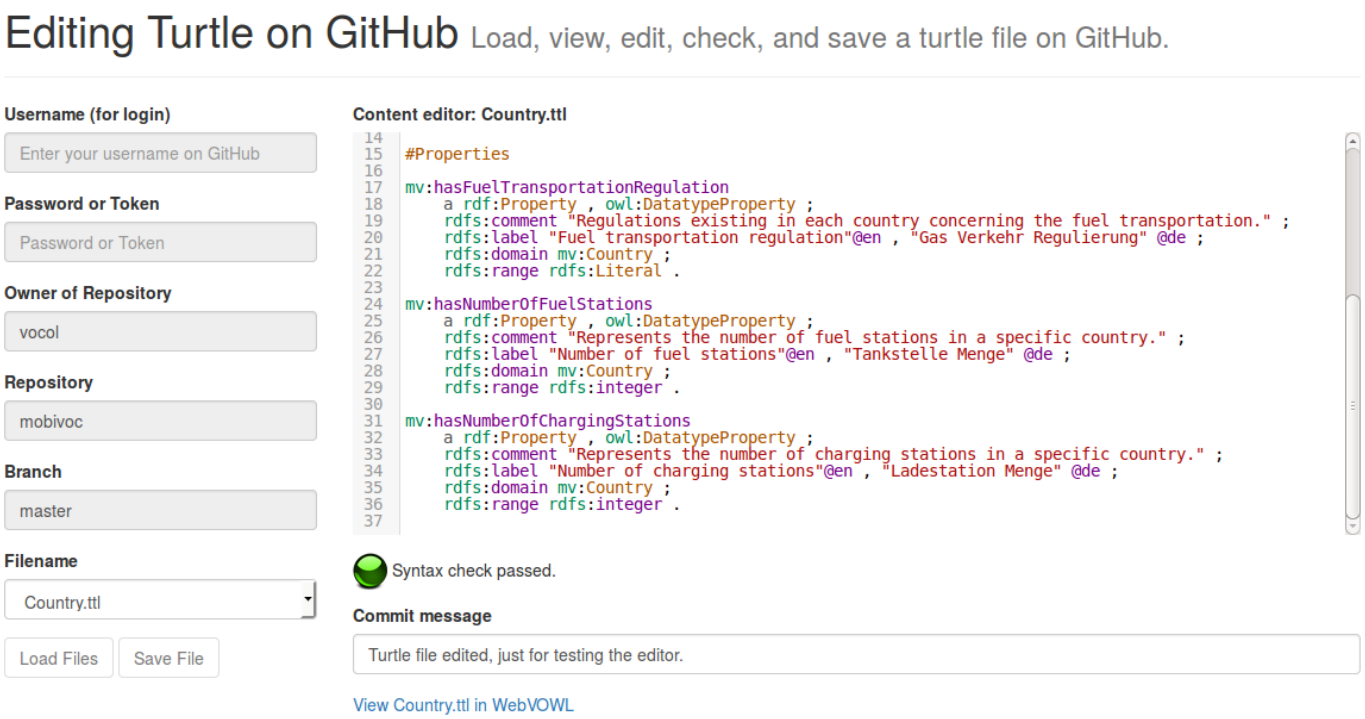

Fig. 2. TurtleEditor front-end

results accordingly. For this, we reused the CodeMirror code editor to allow editing and executing SPARQL queries. The results of the query are presented using a HTML table (See fig. 3).

repository communication The communication to the repository is realized by using the REST interface provided by the GitHub repository. This way the user can specify any given GitHub repository project for TurtleEditor to check out. All files of the repository are filtered depending on the file format ('.ttl') in order to ignore non-vocabulary sources such as *readme* documents. Thus, projects in which the vocabulary consists of multiple modules can be easily edited by switching between each file (using a drop-down list). The form is displayed on the left side of fig. 2.

access control Since the majority of projects hosted on GitHub are open source, it is not always necessary to log in in order to checkout the vocabulary sources. However, committing changes to the repository, as well as reading files from a private project (non-accessible to the public), requires a GitHub account. Instead of logging in with credentials (username and password), we also provide the option to use a generated personal access token to authenticate with the GitHub REST API and thus avoid sharing one's actual credentials with the third party that hosts TurtleEditor. Furthermore, it is possible to select a specific branch to support non-linear vocabulary development projects. By default, the 'master' branch is selected.

IV. Related Work

While there exist multiple approaches to lower the entry barrier for collaborative vocabulary development, our review of related work focuses on the two most popular projects WebProtégé [1] and VocBench [2].

WebProtégé is a web client, which has its own repository service called Protégé Server (also used by Collaborative Protégé [5]), which keeps track of the changes. The client pings the server every 5 seconds and checks if there are any new versions. In case of changes, it provides the diff between the client and server version such that the client can decide what to keep or discard. Furthermore, a chat service is included, as well as the possibility to annotate ontology resources to discuss or propose new changes. WebProtégé requires the vocabularies to be hosted on their own server.

VocBench also provides a web editor for collaborative vocabulary development. Driven by the needs of the shareholders, its user interface is optimized for editing SKOS thesauri. VocBench's features include a full history of changes, a validation and publication workflow and a SPARQL query service. As a backend, it is necessary to deploy an own server including a Apache Tomcat[14] installation with a MySQL database[15] and Ontotext GraphDB[16] installed on a Sesame2 server [17].

[14] http://tomcat.apache.org/
[15] https://www.mysql.com/
[16] http://ontotext.com/products/graphdb/
[17] http://rdf4j.org/

SPARQL editor:

```
PREFIX mv: <http://purl.org/net/mobivoc/>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX schema: <http://schema.org>

SELECT ?fuelname ?price ?stationName
WHERE
{
  ?station a mv:FillingStation .
  ?station rdfs:label ?stationName .
  ?station mv:hasOffer ?fuel .
  ?fuel rdfs:label ?fuelname .
  ?fuel mv:hasPrice ?price .
 }

```

Run Query

fuelname	price	stationName
Aral Super 95	1,57	Aral Tankstelle
Aral Super 95 E10	1,59	Aral Tankstelle
Aral Super Plus 98	1,63	Aral Tankstelle
Aral Super Plus 92	1,60	Aral Tankstelle
Shell Fuel Save 95	1,55	Shell Tankstelle
Shell Fuel Save E10	1,58	Shell Tankstelle
Shell Diesel Fuel Save	1,32	Shell Tankstelle

Fig. 3. SparqlProcessor front-end

However, none of these tools fulfilled our requirements entirely. While history tracking is supported by both solutions, their implementation only covers a subset of the functionality that Git[18] offers and that have proved useful in many collaborative vocabulary development efforts (see section I). Thus, TurtleEditor aims at enhancing the existing features of the many available repository hosting platforms instead of building an isolated solution.

Furthermore, the idea of allowing the user to decide where the vocabulary should be hosted can be considered as a driving factor of our development.

V. Conclusions and Future Work

Triggered by the need for collaborative ontology editors with a low-barrier web frontend and a connection to arbitrary repository backends, we developed open source solution TurtleEditor[19]. We were able to benefit from the existing functionality of multiple mature JavaScript libraries, and there are libraries that provide even further features (support for further syntaxes including JSON-LD, semantic validation and semantic completion, etc.) to be explored.

Current limitations, which we are planning to address soon, include the implementation of an interactive workflow to avoid editing conflicts. Furthermore, we are planning to enhance the syntactic auto-completion: instead of completing vocabulary terms based on hard-coded lists, we plan to integrate the Lookup service provided by Linked Open Vocabularies[20], which is hosted and maintained by the Open Knowledge Foundation. We also plan to auto-generate namespace prefix bindings based on lookups at http://prefix.cc; for example, that, when a user types `foaf:`, the declaration `@prefix foaf: <http://xmlns.com/foaf/0.1/> .`, is added.

Finally, we observed that TurtleEditor can be useful in teaching to familiarize students with the Turtle syntax. It therefore seems promising to assess the potential of TurtleEditor for training one's Turtle and SPARQL skills in a systematic evaluation.

Acknowledgements. This work has been supported by the German Ministry for Education and Research funded project LUCID, and by the European Commission under the Seventh Framework Program FP7 for grant 601043 (http://diachron-fp7.eu).

[18] https://git-scm.com/

[19] https://github.com/vocol/vocol/tree/master/TurtleEditor

[20] http://lov.okfn.org/dataset/lov/

References

[1] T. Tudorache, J. Vendetti, and N. F. Noy, "Web-protege: A lightweight owl ontology editor for the web." in *OWLED*, vol. 432, 2008.

[2] A. Stellato, S. Rajbhandari, A. Turbati, M. Fiorelli, C. Caracciolo, T. Lorenzetti, J. Keizer, and M. T. Pazienza, "Vocbench: A web application for collaborative development of multilingual thesauri," in *The Semantic Web. Latest Advances and New Domains*. Springer, 2015, pp. 38–53.

[3] M. Horridge, T. Tudorache, C. Nyulas, J. Vendetti, N. F. Noy, and M. A. Musen, "Webprotege: a collaborative web based platform for editing biomedical ontologies," *Bioinformatics*, p. btu256, 2014.

[4] B. Leuf and W. Cunningham, *The Wiki Way: Collaboration and Sharing on the Internet*. Addison-Wesley Professional, 2001.

[5] T. Tudorache, N. Noy, J. Vendetti, and T. Redmond, "Collaborative ontology development with protege," Technical report, Standford University, Tech. Rep., 2007.