Implementation of a visual editor for SHACL-based metadata schemas

NOAH VISTE, University of Twente, The Netherlands

The Shapes Constraint Language (SHACL) provides a valuable standard for resource description framework graphs (RDF), allowing ontologies to be formatted and validated in a predictable way. There is a barrier to entry for the creation of SHACL documents as it requires being familiar with the structuring of both RDF and SHACL. A visual editor bridges the gap by showing the relevant SHACL interactions to the user, facilitating the manual creation process. This research aims to investigate the implementation of a visual editor that generates valid SHACL-based metadata schemas, and how it can incorporate intuitive features for seamless user adoption.

Additional Key Words and Phrases: SHACL, RDF, Ontology, RDFLib

1 INTRODUCTION

As humans we have the intrinsic need to categorize, however reality is an all-encompassing system that is unmanageably complex, therefore it has to be broken down into manageable sets of concepts and properties. Ontologies are used to capture the knowledge of portions of reality that are useful for particular purposes, and thereby provide a practical way for the digital domain to interact with the identified structures. One of the main technologies enabling this is RDF, a W3C standard which allows for the representation of interconnected data. SHACL was then introduced to provide restrictions to the graphs in such a way that allows for predictable behavior and validation. According to the W3C, having SHACL be widespread "enhances the functionality and interoperability of the Web"[6].

For a SHACL graph to be considered correct, the data needs to be validated following the constraints declared in the metadata. This is done by writing the SHACL metadata code directly, however this requires a considerable understanding of the SHACL language and it becomes increasingly tedious as the project grows. A user-friendly alternative is to have a visual editor which associates select UI elements with their SHACL code counterparts. This would supplement the creation of SHACL with an alternative that aims to be more intuitive to use. Additionally it would also alleviates the repetitiveness of modifying existing SHACL shapes.

A notable distinction to make is between the metadata and the data itself. The metadata allows for descriptive properties and enforces the defined constraints applied to the data, meaning both the metadata, and the data can be separately edited. The former of which being the priority for this research.

SHACL as a baseline uses SHACL Core, this contains all the base features needed in order to be able to validate SHACL data graphs against the set of conditions. As an extension of the core functionalities, there is SPARQL which adds more complex constraints on top of the existing ones. This research will focus on the Core implementation as SHACL-SPARQL can not exist without it.

The current implementations of editors for SHACL-based metadata schemas are not varied, the focus is often on populating the SHACL data, instead of the creation of the SHACL shapes.

2 PROBLEM STATEMENT

SHACL is a handy language for defining constraint rules to RDF graphs, however it requires a non-negligible amount of repetitive code to define the SHACL constraints and properties. To alleviate some of the workload, the use of a visual editor can simplify the process. This research will explore how such an editor would be implemented, and how it would exist within the current SHACL editor ecosystem.

2.1 Research Questions

This leads to the following research questions:

RQ1: What are the limitations of existing SHACL editors?

RQ2: Which current technologies would allow for the creation of a well-rounded visual SHACL metadata editor?

RQ3: How can a visual editor that generates valid SHACL-based metadata schemas be implemented?

3 RELATED WORK

Compared to other languages, SHACL is relatively new, as such the ecosystem of tools and libraries is still evolving. There are a few existing implementations of SHACL visual editors.

A notable implementation is the open-source Schímatos [11] which handles RDF graphs, and specifically the validation for SHACL-based constraints. The implementation is built in JavaScript with web compatible tools so it is accessible through a hosted website with the ability to be run locally if the servers are experiencing issues. It shows promise, however it is rendered difficult to use from interface issues and un-intuitive behaviour.

Another significant implementation is the Protégé plugin SHACL4P [3]. Protégé is an open-source ontology editor built in Java, the plugin adds support for the SHACL constraint validation as an extension. Protégé exists in both web version and as downloaded software. However as the SHACL support is provided by plugin, it does not work on the web version.

RDFShape is a playground for RDF graphs allowing the visualization of graphs. Built using JavaScript and Scala, it provides an UML-like diagrams of the graph structure, with support of both validation engines, ShEx and SHACL[8].

In a similar vein there is the stand-alone JavaScript RDF editor OntoPad, which focuses on visual representation based on node graphs and provides a node based drag-and-drop SHACL shape builder[1].

CEDAR is an metadata management tool, allowing for the building of metadata schemas and the populating of those schemas[4]. The functionalities provided by the software are extensive, however it is built around its own biomedical metadata ecosystem.

TScIT 37, July 8, 2022, Enschede, The Netherlands

There are also proprietary knowledge graph editors such as Allegrograph, Stardog and TopQuadrant, however these solutions are closed source. It is worth noting that TopQuadrant is behind the SHACL extension DASH, which provides features that allows the nodes to store which datatype should be used when displayed on a user interface.

In the existing implementations, the degree to which the metadata can be edited is of varying quality. Some solutions only provide visualizations, and others allow for editing the code directly without providing a UI layer. Few provide a dedicated UI to allow for the construction of the SHACL constraints.

It is also often unclear exactly what happens behind the scenes when using the interfaces, a notable example of that is CEDAR the ontology editor that exports in JSON-LD. Creating a new template with a single field generates a JSON-LD containing in excess of 100 lines, converted to 16 lines in Turtle format (See Appendix 8.2). The extra metadata is forced on top, and as such can add unwanted bloat and complexity.

4 REQUIREMENTS

It is important to define the requirements a visual editor for SHACL-based metadata schemas needs to meet in order to be able to concretely analyse the existing solutions, as well as provide a target for the implementation.

(1) The editor needs to be able to convert SHACL shapes graphs into the corresponding user interface components.
(2) The editor needs to generate valid SHACL shapes graphs.
(3) The editor needs to recommend field inputs based on the prefixes used.
(4) The editor needs to be intuitive to use by following modern design principles.

In the current ecosystem the RDF editors are either using Java, JavaScript or are closed-source. Nevertheless other programming languages also support the parsing of RDF graphs, notably the RDFLib library in Python[7]. RDFLib has been used along side SPARQL to perform classification on RDF data[5]. The utilities offered by RDFLib provide the basic functionalities needed to interact with the sets of triples as well as traverse through the graph. The Python ecosystem also contains PySHACL which is built on top of RDFLib. The library allows to check the validity of SHACL graphs, such as validating quality criteria for Git repositories[9]. For the purposes of this implementation Python is thus used, as the needed groundwork libraries exist. For the display of the graphical user interface, PYQT5 is used as it is cross-platform and provides all the basic interface features.

Following the W3 SHACL document, the same prefix bindings will be used[6] for easier readability:

Prefix	Namespace
rdf:	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs:	http://www.w3.org/2000/01/rdf-schema#
sh:	http://www.w3.org/ns/shacl#
xsd:	http://www.w3.org/2001/XMLSchema#
ex:	http://example.com/ns#

4.1 SHACL shapes graph to user interface

There are multiple formats an RDF graph can take depending on the expected use cases for the data. Of the many formats to choose from, the notable ones are n-triples, JSON-LD/RDF-XML which all focus on machine processing, and lastly Turtle for human readability. As this research aims to create a tool that is designed specifically for human use, supporting the Turtle format for both importing and exporting is vital. This is also favorable from a loss of information perspective, as the import prefixes defined in the Turtle format for readability are lost when transitioning into one of the other machine-centric formats.

The format of RDF graphs is by nature simple as it is based on a set of triples. Each triple contains the subject, the predicate, and the object. The user interface is determined by the predicates, for instance if it is "<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>" ("rdf:type" using prefixes) which is the equivalent to the letter "a" in the Turtle format, it means that a class is defined.

In order to create the user interface, each triple needs to be iterated over, it is however important to do this in a systematic way, such that the related fields are grouped together. The classes are treated at the top level, while the children are recursively iterated over. For each child an input form for the predicate gets built, and if the object is a literal such as a string or an integer, then the specific input form for that type of literal is created. For example for the boolean, a check box is used that can be either set to true or false. If the object is a BNode which is a reference to another node, it identifies it and recursively repeats the process for that child. If it is not any of the two aforementioned types, it is built with a string input field.

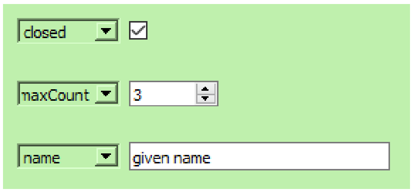
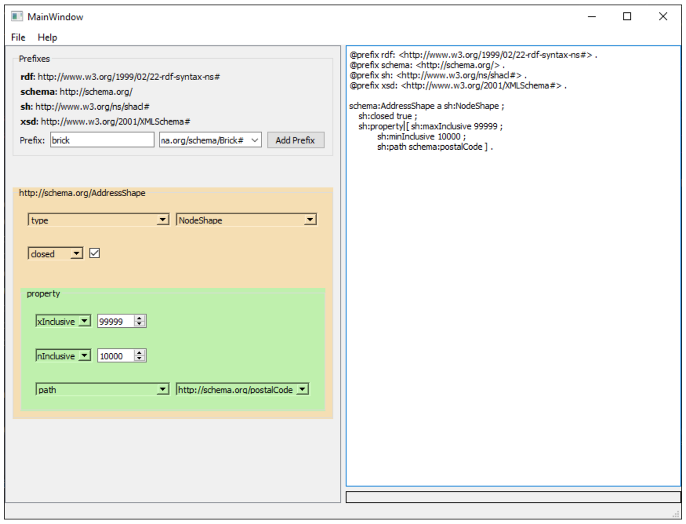


Fig. 1. Simple example of specific fields, from top to bottom: boolean, integer, string

In order to be able to generate the relevant user interface, the editor needs to have access to the SHACL shapes graph. This can be implemented in two ways, either by providing a dedicated load functionality, or by pasting the shapes graph directly into a text edit form. These are not mutually exclusive but for simplicity only the latter is implemented.

Therefore the feature of converting from the shapes graph to a user interface is implementable by taking the graph pasted into the text edit form and parsing it through the RDF library after every edit or on saving. Ensuring the user interface responds to the new loaded graph. The benefits of this solution is the ease of implementation, however it is liable to bring performance issues. The feature can be improved by either accounting for changes, and only acting upon those. This on the other hand requires an in-depth and flexible link between the user interface and graph. A middle ground is to separate the classes, thus if one of the classes gets changed, none of the other would need to be reloaded, avoiding unnecessary processing.



rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
schema: http://schema.org/
sh: http://www.w3.org/ns/shacl#
xsd: http://www.w3.org/2001/XMLSchema#
Prefix: brick | na.org/schema/Brick#

type	NodeShape
closed	✓
xInclusive	99999
nInclusive	10000
path	http://schema.org/postalCode

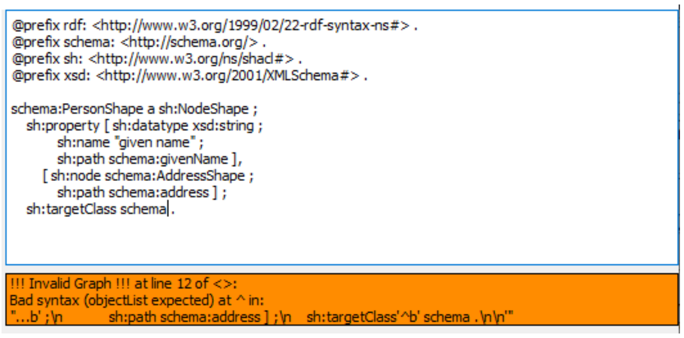
```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix schema: <http://schema.org/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

schema:AddressShape a sh:NodeShape ;
  sh:closed true ;
  sh:property [ sh:maxInclusive 99999 ;
      sh:minInclusive 10000 ;
      sh:path schema:postalCode ] .
```

Fig. 2. Simple example of generated UI from the text input field

Figure 2 above is unable to show show the dynamic updating of the user interface, however when writing in a new sh:minInclusive value on the right side, the left side spin box (integer input field with buttons to increment and decrement) immediately updates with the new values.

Another benefit to this approach is that by loading the SHACL code into the RDF library it ends up verifying that the structure is correct, and in the case of invalid code it provides an informative error message. As the user is editing the text, the left side user interface does not update until it receives a valid shapes graph.



```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix schema: <http://schema.org/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

schema:PersonShape a sh:NodeShape ;
   sh:property [ sh:datatype xsd:string ;
         sh:name "given name" ;
         sh:path schema:givenName ],
      [ sh:node schema:AddressShape ;
         sh:path schema:address ] ;
   sh:targetClass schema.
```

```
!!! Invalid Graph !!! at line 12 of <>:
Bad syntax (objectList expected) at ^ in:
"...b' ;\n          sh:path schema:address ] ;\n    sh:targetClass'^b' schema .\n\n'"
```

Fig. 3. Error message when removing ":person" from "schema:person", the rest of the user interface is cropped away for readability

The transition discussed in this section is the key missing component in all of the other editor implementations, with the exception of Schímatos and OntoPad. Schímatos uses the same type of user interface as this implementation, focusing on fields. On the other hand OntoPad uses a node system with drawn links between the related elements. However as SHACL graphs are usually represented using Turtle, having the formatting match between the user interface and the graph provides a clear and intuitive conversion. Additionally by having the fields built recursively, it makes the hierarchy, and thus the structure of the graph easier to identify.

4.2 Generate SHACL shapes graph

Transitioning from the user interface to a SHACL shapes graph requires significantly more work than parsing a graph. Each user interface field needs to be connected to the graph triple counterpart, such that when the field is modified, the graph is updated as well. In the case of RDFLib this is non-trivial as many of the graph entities are immutable. In order to bypass this restriction a wrapper class is added ontop (see Fig.4) in such a way that it can make alterations to the graph. Afterwards each generated editable field is assigned a signal that is emitted when a change occurs, this causes the wrapper to update the values, and tries to export the graph in Turtle format to the text input on the right half.

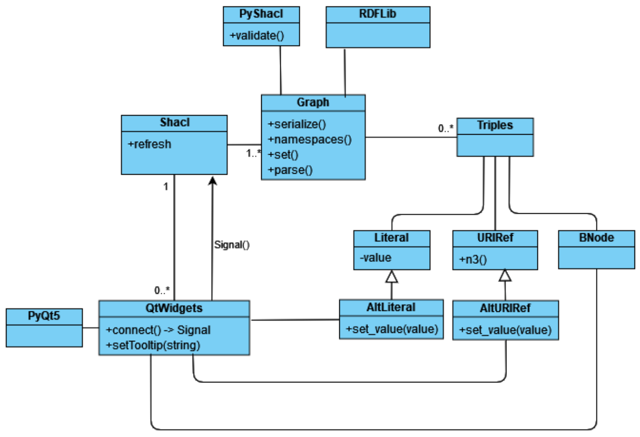


Fig. 4. Class diagram showing simplified structure

Additionally PySHACL verifies the graph to ensure that it conforms to the SHACL requirements.

4.3 Input recommendation

As a baseline whenever adding a new triple, the subject, the predicate and object can be freely chosen, however in order to provide ease of use filtering is needed. For instance "sh:maxLength" requires an "xsd:integer" as datatype, in which case the object field should be limited to integers only as shown in the shapes graph to user interface section. Similarly the severity predicate should only have 3 options available "sh:Info", "sh:Warning" and "sh:Violation". As there are an extensive amount of predicates and classes, by using the SHACL itself loaded into the RDF library, the fields can be filtered by querying the SHACL graph, instead of having to manually declare all the conditions.

The recommendations serves two purposes, it both helps find the option that is wanted as well as providing a basic auto-completion

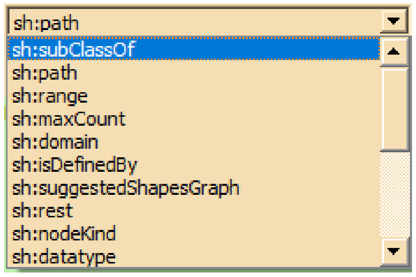
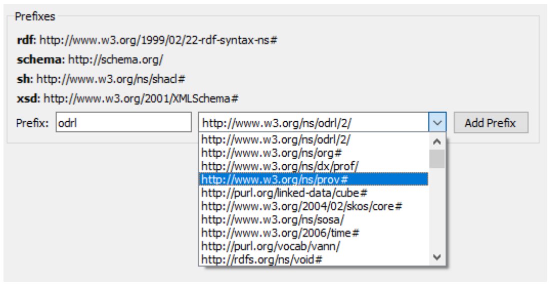


Fig. 5. Simple drop-down hint menu

for manual typing. This helps reduce the barrier to entry as it helps guide the user into making a valid decision.

The same procedure works for the prefixes. In order to receive a list of all the prefixes and the commonly used abbreviations, the namespace lookup website prefix.cc allows for querying. In order to avoid unnecessary querying, the list can be downloaded and parsed through RDFLib. When used this way however, the list might become outdated.


Prefixes

rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#
schema: http://schema.org/
sh: http://www.w3.org/ns/shacl#
xsd: http://www.w3.org/2001/XMLSchema#

Prefix: odrl http://www.w3.org/ns/odrl/2/

Add Prefix
http://www.w3.org/ns/odrl/2/
http://www.w3.org/ns/org#
http://www.w3.org/ns/dx/prof/
http://www.w3.org/ns/prov#
http://purl.org/linked-data/cube#
http://www.w3.org/2004/02/skos/core#
http://www.w3.org/ns/sosa/
http://www.w3.org/2006/time#
http://purl.org/vocab/vann/
http://rdfs.org/ns/void#

Fig. 6. Simple prefix drop-down hint menu

Each prefix that is added to the prefix list can be fetched and then used to extend the set of hints provided when filling out the forms, as shown in Fig 5.

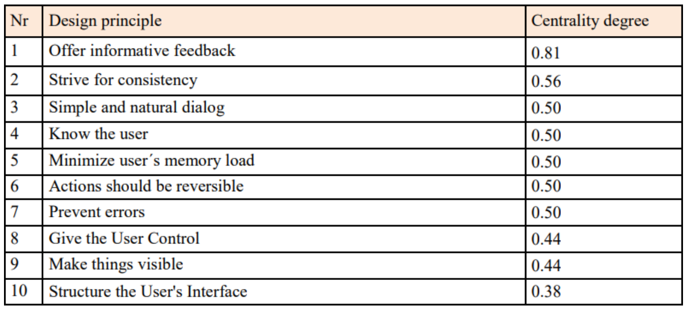
OntoPad partially covers this, it exposes the class list and properties, among others, which can be click and dragged onto the shape nodes. However it does not expose object filtering when connecting nodes, as well as not having an easily accessible way to modify the object.

4.4 Intuitive to use

In order to have an intuitive editor to use, it is important to define "intuitive". Ruiz et al. (2020) compiled a list of 50 design principles with the highest centrality degree[10].

An issue with existing SHACL editor solutions is the schism between the user interface and the resulting graphs. As the SHACL shape graph is being built, the text format equivalent is hidden, usually only shown during export. As the Turtle format is inherently easy to read and understand, having both the user interface and the Turtle equivalent shown side by side would help provide concrete feedback for the user actions.

Emphasizing the "What You See Is What You Get" approach allows for more informed decisions and gives a clearer understanding of



Nr	Design principle	Centrality degree
1	Offer informative feedback	0.81
2	Strive for consistency	0.56
3	Simple and natural dialog	0.50
4	Know the user	0.50
5	Minimize user´s memory load	0.50
6	Actions should be reversible	0.50
7	Prevent errors	0.50
8	Give the User Control	0.44
9	Make things visible	0.44
10	Structure the User's Interface	0.38

Fig. 7. The top 10 of the 50 highest ranked design principles relative to the centrality degree[10]

the process. This covers the most important design principle of offering informative feedback.

One of the shortcomings of using a recursive field user interface instead of a node system, is that the readability suffers greatly when multiple layers deep. To combat this, each layer receives its own color from a gradient to clearly delineate which fields correspond to which groups (see Fig 8). To add a triple to a group, it is achieved by right clicking inside the colored rectangle in which it is found.

Of the only two relevant existing implementations, Schímatos does not provide informative feedback for the actions, it is unclear which process is taking place, this is exacerbated by the structure of the user interface. Additionally there are unforeseeable errors that cause the progress to be lost.

OntoPad and Schímatos also have visibility issues, as they have several submenus, and certain menus that only show up as needed which makes it hard to make a decision when the user is not aware that they exist.

5 LIMITATIONS

Python despite its flexibility and many use cases, is unable to run natively in the browser. This is a significant limitation as the ease of access becomes severely hindered without sacrificing functionality. Therefore choosing a language that easily support web could be beneficial.

The editor does not online query the prefixes, it is therefore unable to provide recommendations for prefixes that are not locally defined.

Following the Turtle format also introduces some limitations. As the graph structure grows and becomes more complex, it can be hard to visualize the inter-linking of the nodes. This is due to the tree structure, however a node based system provides the freedom to better show how the graph is connected.

Non-validating property shape characteristics such as "sh:name", "sh:description", "sh:order", "sh:group", and "sh:defaultValue" provide extra metadata information that is not used for logic in the graph structure. The order property can be given to each property, "if present at property shapes, the recommended use of sh:order is to sort the property shapes in an ascending order, for example so that properties with smaller order are placed above (or to the left) of properties with larger order"[6]. This can help showing the

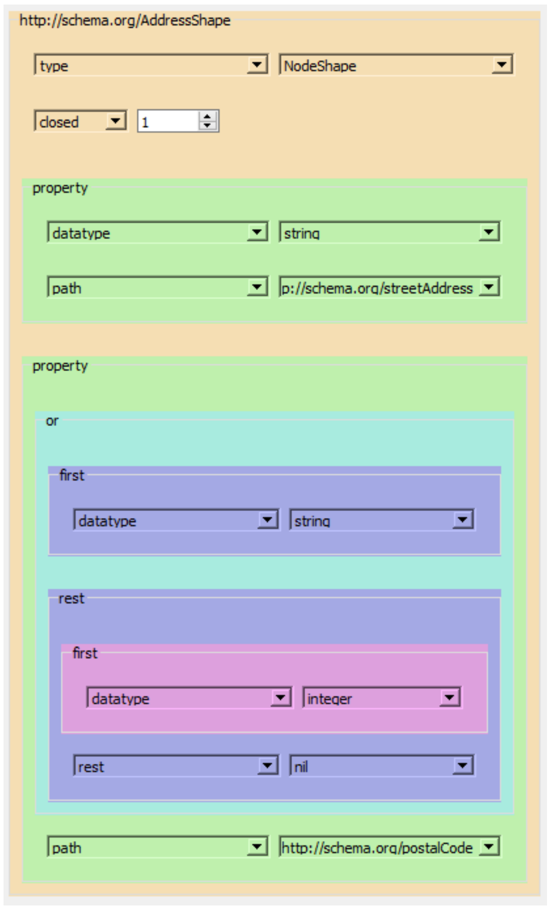

http://schema.org/AddressShape
type NodeShape
closed 1
property
datatype string
path p://schema.org/streetAddress
property
or
first
datatype string
rest
first
datatype integer
rest nil
path http://schema.org/postalCode

Fig. 8. Visible gradient from a 5 layers deep AddressShape

user interface in a consistent way such that fields do not move unpredictably when making changes.

6 CONCLUSION

This research identifies the existing editor solutions and compares their functionalities against the requirements of SHACL-based metadata schema editing. Creating a SHACL shapes graph requires a good understanding of SHACL to get started, this barrier to entry only hinders the widespread adoption that the W3C is looking for. However by guiding the user with curated selection options and by providing immediate feedback, it simplifies the adoption process. Aside from being more accessible, those qualities can also help streamline the creation process.

Only two editors pass the main functionality requirement that the SHACL shapes graphs can be converted into the user interface equivalent. Schímatos provides the needed functionalities, however it lacks intuitiveness and the user interface becomes difficult to use when the amount of field to fill in grows. OntoPad on the other hand provides a different experience with node based building, however for metadata schema editing it adds a layer of complexity.

This research shows a process for creating an editor that interconnects the user interface with the SHACL directly. It follows the design principles that aim to provide a usable user experience, such as dedicated input fields. The end result is not extensive however it showcases the functionalities.

7 FUTURE WORK

As the implementation stands, SPARQL definitions are accepted, however they are only editable as strings. This can be improved by full SPAQRL support allowing for the structured editing of the queries. However it adds a layer of complexity to the user interface.

DASH is an extension of SHACL that provides new constraints and target types among other features. Parts of the extension introduces SHACL shape definitions that aim to help create user interfaces. It provides more specific functionalities than the basic ones found natively in SHACL. If implemented this feature would add more input types, and thus allow for better control of the user experience.

Auto-completion is a invaluable quality of life feature which greatly streamlines the tediousness of filling in input boxes. Shacled Turtle explores two different filtering strategies, and conclude that a naive approach of displaying all the suggestion is equivalent to their unique solution that filters out suggestions[2]. They acknowledge different filtering options that could provide better results, such as recommending the entire triplet instead of one field at a time.

8 APPENDIX

8.1 Gitlab

https://gitlab.utwente.nl/s2343169/shaclomatic

8.2 CEDAR empty export

Converted from JSON-LD to Turtle using EasyRDF. Formatted for readability.

```
@prefix ns0: <http://open-services.net/ns/core#> .
@prefix bibo: <http://purl.org/ontology/bibo/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix ns1: <http://purl.org/pav/> .
@prefix schema: <http://schema.org/> .

<https://repo.metadatacenter.org/templates/x>
  a <https://schema.metadatacenter.org/core/Template> ;
  ns0:modifiedBy <https://metadatacenter.org/users/x> ;
  bibo:status "bibo:draft"^^xsd:string ;
  ns1:createdBy <https://metadatacenter.org/users/x> ;
  ns1:createdOn "x"^^xsd:dateTime ;
  ns1:lastUpdatedOn "x"^^xsd:dateTime ;
  ns1:version "0.0.1"^^xsd:string ;
  schema:description ""^^xsd:string ;
  schema:name "Empty-Example"^^xsd:string ;
```

```
    schema:schemaVersion "1.6.0"^^xsd:string .
```

REFERENCES

[1] Natanael Arndt, André Valdestilhas, Gustavo Publio, Andrea Cimmino Arriaga, Konrad Höffner, and Thomas Riechert. 2021. A Visual SHACL Shapes Editor Based On OntoPad.. In *SEMANTiCS Posters&Demos*.

[2] Julian Bruyat, Pierre-Antoine Champin, Lionel Médini, and Frederique Laforest. 2022. Shacled turtle: Schema-based turtle auto-completion. In *Visualization and Interaction for Ontologies and Linked Data 2022*, Vol. 3253. 2–15.

[3] Fajar J. Ekaputra and Xiashuo Lin. 2016. SHACL4P: SHACL constraints validation within Protégé ontology editor. In *2016 International Conference on Data and Software Engineering (ICoDSE)*. 1–6. https://doi.org/10.1109/ICODSE.2016.7936162

[4] Rafael S. Gonçalves, Martin J. O'Connor, Marcos Martínez-Romero, Attila L. Egyedi, Debra Willrett, John Graybeal, and Mark A. Musen. 2017. The CEDAR Workbench: An Ontology-Assisted Environment for Authoring Metadata that Describe Scientific Experiments. In *The Semantic Web – ISWC 2017*, Claudia d'Amato, Miriam Fernandez, Valentina Tamma, Freddy Lecue, Philippe Cudré-Mauroux, Juan Sequeda, Christoph Lange, and Jeff Heflin (Eds.). Springer International Publishing, Cham, 103–110.

[5] Rupal Gupta and Sanjay Kumar Malik. 2022. A classification using RDFLIB and SPARQL on RDF dataset. *Journal of Information and Optimization Sciences* 43, 1 (2022), 143–154. https://doi.org/10.1080/02522667.2022.2039461 arXiv:https://doi.org/10.1080/02522667.2022.2039461

[6] Holger Knublauch and Dimitris Kontokostas. 2017. Shapes Constraint Language (SHACL), W3C Recommendation. *World Wide Web Consortium* (2017).

[7] D Krech. 2006. Rdflib: A python library for working with rdf. *Online https://github.com/RDFLib/rdflib* (2006).

[8] Jose Emilio Labra Gayo, Daniel Fernández-Álvarez, and Herminio Garcıa-González. 2018. RDFShape: An RDF playground based on Shapes. In *Proceedings of ISWC*.

[9] Leon Martin and Andreas Henrich. 2022. Specification and Validation of Quality Criteria for Git Repositories using RDF and SHACL. (2022).

[10] Jenny Ruiz, Estefanía Serral, and Monique Snoeck. 2021. Unifying Functional User Interface Design Principles. *International Journal of Human–Computer Interaction* 37, 1 (2021), 47–67. https://doi.org/10.1080/10447318.2020.1805876 arXiv:https://doi.org/10.1080/10447318.2020.1805876

[11] Jesse Wright, Sergio José Rodríguez Méndez, Armin Haller, Kerry Taylor, and Pouya G. Omran. 2020. Schímatos: A SHACL-Based Web-Form Generator for Knowledge Graph Editing. In *The Semantic Web – ISWC 2020*, Jeff Z. Pan, Valentina Tamma, Claudia d'Amato, Krzysztof Janowicz, Bo Fu, Axel Polleres, Oshani Seneviratne, and Lalana Kagal (Eds.). Springer International Publishing, Cham, 65–80.