Shape Designer for ShEx and SHACL Constraints*

Iovka Boneva[1], Jérémie Dusart[2], Daniel Fernández Álvarez[3], and Jose Emilio Labra Gayo[3]

[1] Univ. Lille, CNRS, Centrale Lille, Inria, UMR 9189 - CRIStAL - Centre de Recherche en Informatique Signal et Automatique de Lille, F-59000 Lille, France
[2] Inria, France
[3] University of Oviedo, Spain

Abstract. We present Shape Designer, a graphical tool for building SHACL or ShEx constraints for an existing RDF dataset. Shape Designer allows to automatically extract complex constraints that are satisfied by the data. Its integrated shape editor and validator allow expert users to combine and modify these constraints in order to build an arbitrarily complex ShEx or SHACL schema.

Keywords: RDF validation, SHACL, ShEx, Wikidata, LOD quality

1 Introduction

The Shape Constraint Language (SHACL) and Shape Expressions Language (ShEx) are gaining popularity for asserting quality of RDF datasets, but also for describing their structure. However, the construction of SHACL or ShEx schemas remains a difficult problem. It requires to master different tools and languages and swap between them in order to complete a schema construction task: the syntax and semantics of the constraint language, the existing validation APIs or tools, query languages, or other means of exploring the data. Shape Designer integrates all these functionalities within a single graphical user interface. It can help non expert users to grasp the structural characteristics of an RDF dataset thanks to the automatic construction of valid constraints. Expert users on the other hand can use *constraint patterns* to parameterize the automatic extraction algorithm, use the schema editor to build a complex schema, and test the quality of the schema or the data thanks to the integrated validator.

In this demo we present two use cases of Shape Designer. The tool and user documentation are available on `https://gitlab.inria.fr/jdusart/shexjapp`.

2 Tool Overview

A Shape Designer project is associated with an RDF graph **G** and a ShEx or SHACL schema under construction **S**. The language to use for the schema,

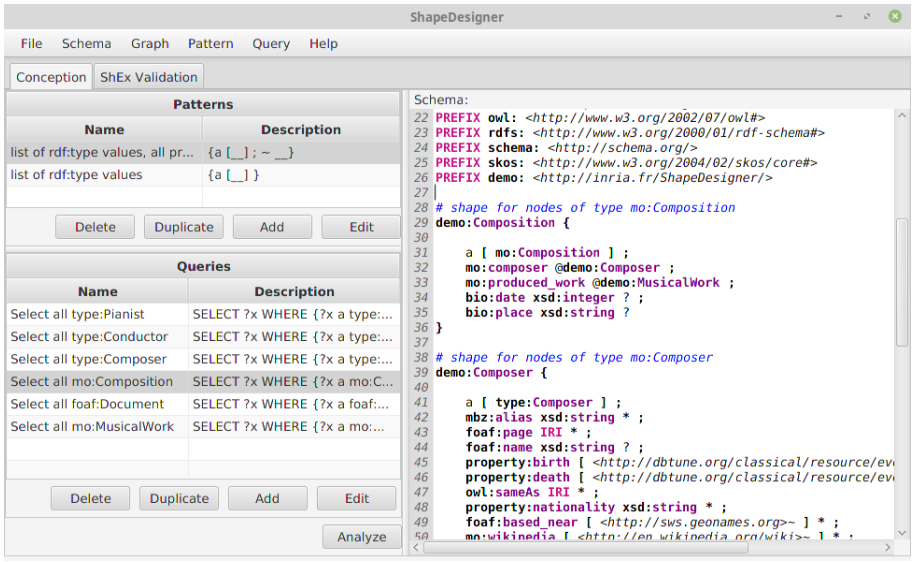

Fig. 1. A schema under development in Shape Designer

ShEx or SHACL, is fixed at project creation. The GUI of Shape Designer is shown in Fig. 1. It includes the following panels: (1) a list of shape patterns; (2) a list of node selection queries; (3) an editing area containing the schema under construction **S**; (4) a validation view (not shown in Fig. 1).

The main functionality of Shape Designer is the automatic construction of a (ShEx or SHACL) shape constraint. For a given node selection query Q and a shape pattern P, it creates a shape constraint $constr(Q, P)$ that has a structure as indicated by pattern P and that is satisfied by all the nodes of **G** in the result of query Q. The constraint thus obtained can be added to the schema under construction, or simply used to gather information about the data.

3 Use Case : Constraint for every rdf:type

We demonstrate how to construct a recursive ShEx schema **S** that has one constraint for every rdfs:Class used in **G** and accounts for the references between types. We use a dataset with uniform structure from DBTunes[4].

Create the project. At project creation we choose the shape language, here ShEx, and indicate to the tool where to find the graph: in a local file, a local RDF4J database, or through a SPARQL endpoint. A new project comes with a set of predefined useful patterns. For instance, pattern P_1 = { rdf:type [__] ; ~__ } is predefined and indicates that the constraint to be constructed should contain the list of possible values ([__]) for property rdf:type, and a datatype or

[4] downloaded from `http://dbtune.org/classical/` on June 16, 2019.

```
demo:Composition {                    :Shape_QComp_P2 {
   a [ mo:Composition ]      ;           mo:composer {
   mo:composer [ composer:~ ];              a [ type:Composer ] }     ;
   mo:produced_work IRI  *   ;           mo:produced_work {
   bio:date  xsd:integer ?   ;              a [ mo:MusicalWork ] } *
   bio:place xsd:string  ?   }         }
```

Fig. 2. Automatically generated shapes.

node kind value constraint (_) for all other properties (~). We also ask the tool to construct queries that select all nodes of some class, for all the classes that appear as objects of rdf:type in the graph. For instance, query Q_{Comp} = SELECT ?x WHERE {?x a mo:Composition} is generated by the tool. Note that the prefixes defined in the graph can be used in queries and in patterns.

Automatic construction of shapes. Now, given P_1 and Q_{Comp}, the tool inspects the graph and constructs $constr(Q_{\text{Comp}}, P_1)$ shown on the left hand side of Fig. 2. Remark that the cardinality of each triple constraint was automatically inferred from the data, among four possible cardinality constraints that are *exactly one*, *optional* (?), *at least one* (+), and *any number* (*). We can choose to add $constr(Q_{\text{Comp}}, P_1)$ to the schema **S** under construction. By repeating the construction of $constr(Q_C, P_1)$ for all classes C, a novice user can build a schema that models the data without need of mastering the shape constraint language. For instance, we create shapes demo:Composer and demo:MusicalWork for the nodes with classes type:Composer and mo:MusicalWork, respectively.

Validation. At any time the user can validate graph **G** against the schema under construction **S**. The validation result is shown in a list and allows to explore the neighbourhood of nodes to help understand validation errors.

Shape patterns to explore the structure of the data. Automatic shape construction can be used to gather information about the data. Consider pattern P_2 = { mo:~ { rdf:type [_] } }. It indicates that we are interested only in properties with namespace mo:, and we want to retrieve the lists of rdf:types of their values. The shape $constr(Q_{\text{Comp}}, P_2)$ is presented on the right-hand side of Fig. 2 and it shows that the values of property mo:composer have class type:Composer, and the values of property mo:produced_work have class mo:MusicalWork.

Editing the schema. Automatically inferred schemas use only subsets of the ShEx and SHACL languages. For instance, they never use shape references. We can create complex schemas by editing the schema under construction. For instance, using the information gathered from $constr(Q_{\text{Comp}}, P_2)$ we can introduce shape references in demo:Composition in Fig. 2 as follows.

```
mo:composer      @demo:Composer ;
mo:produced_work @demo:MusicalWork *
```

4 Use Case : Explore Wikidata

Wikidata is (partially) crowd-sourced, thus highly heterogeneous. This is an essential difference compared to the first use case in which nodes with same rdf:type have very similar properties. Shape Designer can be used to understand the structure of Wikidata entries and assert the quality of such entries for a particular application in mind. We assume the reader familiar with the basic concepts of Wikidata (direct properties, property statements, qualifiers).

Shape Designer offers special support for Wikidata in the form of Wikidata project type that comes with predefined prefixes and patterns. Shape construction and validation are performed by querying the Wikidata SPARQL endpoint, therefore we put a limit on the number of query results. Validating the whole Wikidata set or even all entities of given type is computationally prohibitive. In this context the most useful feature of Shape Designer are patterns, as they allow to focus on a particular kind of information in Wikidata. For instance, a simple pattern for direct properties allows to get an idea on which properties can be expected for entities of given type. More complex patterns can account for the presence of absence of provenance information.

5 Conclusion

We have presented Shape Designer, a versatile tool for constructing shape schemas or exploring RDF datasets designed to be usable by novices and experts. Shape Designer uses ideas similar to several prototype tools for extracting ShEx or SHACL constraints from RDF graphs or from Wikidata [1,2,3,4,5]. The originality of Shape Designer lies in shape patterns that provide a general and succinct way to parametrize the automatic shape construction, and its user interface that integrates schema editor, validator, and a way to explore the data.

Shape Designer is being actively developed, in particular by adding more functionality for exploring Wikidata. In the near future we plan to extend shape patterns and the schema construction algorithm so that even more complex schemas could be built automatically, including recursive shapes.

Acknowledgments This work was partially funded by a grant from CPER Nord-Pas de Calais/FEDER DATA Advanced data science and technologies 2015-2020, by the ANR project DataCert ANR-15-CE39-0009, by the Spanish Ministry of Economy and competitiveness (Society Challenges: TIN2017-88877-R), and by the "Severo Ochoa" research program (BP17-88).

References

1. Fernández-Álvarez, D., García-González, H., Frey, J., Hellmann, S., Labra Gayo, J.E.: Inference of Latent Shape Expressions Associated to DBpedia Ontology. In: International Semantic Web Conference (2018)
2. Labra Gayo, J.E., Fernández-Álvarez, D., García-González, H.: RDFShape: An RDF playground based on Shapes. In: Proceedings of ISWC (2018)
3. Principe, R.A.A., Spahiu, B., Palmonari, M., Rula, A., De Paoli, F., Maurino, A.: ABSTAT 1.0: Compute, Manage and Share Semantic Profiles of RDF Knowledge Graphs. In: European Semantic Web Conference. pp. 170–175 (2018)
4. Spahiu, B., Maurino, A., Palmonari, M.: Towards Improving the Quality of Knowledge Graphs with Data-driven Ontology Patterns and SHACL. In: WOP@ISWC
5. Werkmeister, L.: Schema Inference on Wikidata. Master Thesis (2018)