Mission Editorial Committee Process and Structure Code4Lib

Search

Issue 45, 2019-08-09

Design reusable SHACL shapes and implement a linked data validation pipeline

In July 2017, W3C published SHACL as the standard to validate RDF. Since then, data modellers have the possibility to provide validation services based on SHACL shapes together with their models, however there are considerations to be taken in account when creating them. This paper aims to list such considerations and shows an example of a validation pipeline to address them.

by Emidio Stani

Introduction

With the increasing demand of data to be provided to citizens, organizations have the challenge to make their processes more efficient so that decisions can be made fast, while maintaining and even increasing confidence in their data.

Therefore, organizations have the need to structure their data properly as the risk to have poor data quality might incur in high costs [1].

Thus, organizations are encouraged to invest into applying metadata that plays a fundamental role beyond classifying data, as data needs to be transformed, integrated, and transmitted. However, like data, metadata needs to be harvested, standardized and validated.

In the case of linked data, expressed as RDF, the harvesting phase has already become an important step implemented at large scale, see for example the European Data Portal (https://www.europeandataportal.eu/en/homepage) that provides access to 81 harvested catalogues for a total of 892,354 datasets.

This is possible only by defining a common data model that allow describing catalogues and datasets in a uniform way, as is the case of DCAT-AP for the European Data Portal (https://www.europeandataportal.eu/en/providing-data/goldbook/preparing-data).

Once a standard is defined, a validation service can built. Such service will be based on violation rules, which in turn will depend on the data model.

In RDF, it is already possible to validate data by means of querying it via the SPARQL language but as of July 2017, W3C released the SHACL specification (https://www.w3.org/TR/shacl/) by which data modelers can provide SHACL shapes together with their data models so that users can reuse them to validate their data.

At the European level, the European Commission developed a few application profiles such as DCAT-AP (https://ec.europa.eu/isa2/solutions/dcat-application-profile-data-portals-europe_en) and CPSV-AP (https://ec.europa.eu/isa2/solutions/core-public-service-vocabulary-application-profile-cpsv-ap_en) (and some Core Vocabularies such as Person, Organization, etc.) for which the physical data models have been implemented in RDF. Each release of the application profiles is accompanied with the respective SHACL shapes.

In the past, the European Commission developed a SPARQL validator for DCAT-AP (http://dcat-ap.semic.eu/dcat-ap_validator.html) giving a user-friendly way to validate catalogue and dataset descriptions and where the SPARQL query is the union of several SPARQL queries checking the presence of properties/relations described in the DCAT-AP data model.

Currently, the European Commission is providing a testbed as a service (https://www.itb.ec.europa.eu/shacl/swagger-ui.html) and product that allows users to check conformance of the application profiles by uploading files, provide a URL or upload multiple files.

At national or local level, we see the case of the Flemish Government (https://data.vlaanderen.be/) which developed a data model, called OSLO2 (https://overheid.vlaanderen.be/oslo-wat-is-oslo2), partially based on the European data models, and aimed to standardize the data in the government of Flanders. The Flemish Government implemented as well a tool chain to generate specifications and documentation from the UML class diagram of OSLO2, among them the SHACL shapes for each application profile (https://data.vlaanderen.be/shacl/).

On top of that, the Flemish Government released the so-called OSLO2 validator that provides a way to test data against the OSLO2 SHACL shapes (https://data.vlaanderen.be/shacl-validator/) where users can upload a file or just provide the URL.

Both approaches, the European and the Flemish Government, depend on:

- the way SHACL shapes are defined, which are not modular enough to be reused, forbidden and evaluated;
- the data provided, but there might be the need to enrich such data before validating by means of inferencing or business rules;
- the interface which oblige users to have few possibilities to validate data.

Introduction to SHACL

SHACL (Shapes Constraint Language) is a standard released by W3C in 2017 which aims to validate RDF data by means of so-called *shapes*. Such shapes can be described in RDF itself, therefore allowing RDF store to be able to store them next to the RDF data contrary to SPARQL which has been defined with its own language.

Each shape is defined by 2 parts:

- target, to indicate the target (RDF class, specific RDF nodes, etc.) of the shape;
- constraints, to be applied to the target (cardinalities, range of values, etc.)

In the example below (see Listing 1), there is an example of a shape defined as ex:PersonShape which has as target all the classes (sh:targetClass) of type ex:Person.

Further the shape defines 2 constraints on the property ex:name for which:

1. the cardinality must be minimum 1 (that is a way to define mandatory property);
2. The data type must be a string.

```
ex:PersonShape
  a sh:NodeShape ;
  sh:targetClass ex:Person ;  # Applies to all persons
  sh:property [               # blank node
    sh:path ex:name ;         # constrains the values of ex:name
    sh:minCount 1 ;
    sh:datatype xsd:string ;
  ] .

```

Listing 1. example of SHACL shape

It is worth to note that the constraints are defined within a blank node (https://en.wikipedia.org/wiki/Blank_node).

Further, SHACL defines 2 types of shapes:

- NodeShapes, which allow to specify constraints about a focus node;
- PropertyShapes, which allow to specify constraints about a property and the values of a path of a node.

For example, another way to describe the shape described in Listing 1 is to declare the constraint explicit with a PropertyShape (ex:PersonShape-name), which contains the same constraints, and it is linked from the NodeShape as illustrated in Listing 2.

```
ex:PersonShape-name
  a sh:PropertyShape ;
  sh:path ex:name ;
  sh:minCount 1 ;
  sh:datatype xsd:string .

ex:PersonShape
  a sh:NodeShape ;
  sh:targetClass ex:Person ;
  sh:property ex:PersonShape-name .

```

Listing 2. example of SHACL PropertyShape

This document invites the reader to use PropertyShapes as much as possible as described within the next section.

Defining reusable SHACL Shapes

As mentioned earlier, SHACL shapes are described in RDF itself so they can be managed in the same way as RDF data, however they can be described in different ways affecting their reusability.

As an example OSLO2 SHACL shapes (and DCAT-AP shapes) are defined with properties described as blank nodes e.g.:

```
<https://data.vlaanderen.be/shacl/dienstencataloog#PubliekeDienstverleningShape>
  a sh:NodeShape ;
  sh:targetClass <http://purl.org/vocab/cpsv#PublicService> ;
  sh:property [   # blank node
    sh:name "heeftParticiperende" ;
    sh:description "Participatie (van een agent) aan de publieke dienstverlening." ;
    sh:path <http://data.europa.eu/m8g/hasParticipation> ;
    sh:class <http://data.europa.eu/m8g/Participation> ;
  ] ;

  sh:property [   # blank node
    sh:name "beschrijving" ;
    sh:description "Beschrijving vd publieke dienstverlening." ;
    sh:path <http://purl.org/dc/terms/description> ;
    sh:datatype <http://www.w3.org/1999/02/22-rdf-syntax-ns#langString> ;
    sh:minCount 1 ;
    sh:maxCount 1 ;
  ] .


```

Listing 3. example of a OSLO2 NodeShape with blank nodes

The problem of the definition of blank nodes is that other SHACL shapes, described in the same data model or in other data models, cannot reuse such properties in contrast with the spirit of Semantic web where reusability (by interlinking) is desired.

A more concise way to describe such SHACL shapes would be declaring explicitly the PropertyShapes and referencing to them. Listing 4 shows 2 PropertyShapes and 2 NodeShapes defined within the cpsv namespace (the reader can find a more detailed example on https://github.com/catalogue-of-services-isa/CPSV-AP/blob/master/releases/2.2.1/CPSV-AP_shacl_shapes%20v2.2.1.ttl):

```
cpsv:PublicServiceShape-hasParticipation
  a sh:PropertyShape ;
  sh:path cv:hasParticipation ;
  sh:class cv:Participation ;
  sh:severity sh:Violation .

cpsv:hasMin1Max1Shape-description
  a sh:PropertyShape ;
  sh:path dct:description ;
  sh:minCount 1 ;
  sh:maxCount 1 ;
  sh:nodeKind sh:Literal ;
  sh:severity sh:Violation .

cpsv:PublicServiceShape
  a sh:NodeShape ;
  sh:targetClass cpsv:PublicService ;
  sh:property cpsv:PublicServiceShape-hasParticipation ;
  sh:property cpsv:hasMin1Max1Shape-description .

# shape reusing cpsv:hasMin1Max1Shape-description
cpsv:ParticipationShape
  a sh:NodeShape ;
  sh:targetClass cv:Participation ;
  sh:property cpsv:hasMin1Max1Shape-description ;
  sh:property cpsv:hasMin1Max1Shape-identifier ;
  sh:property cpsv:ParticipationShape-role .
```

Listing 4. example of a CPSV-AP shapes reusing propertyShapes

In Listing 4, we can see, for example, that the PropertyShape hasMin1Max1Shape-description is reused by the PublicServiceShape and the ParticipationShape and requires a description property (dct:description) while the PublicServiceShape-hasParticipation is only used by PublicServiceShape requiring a Participation class (cv:Participation).

In the case of DCAT-AP, SHACL shapes are described by reusing the same classes (dcat:Catalog) declared in DCAT-AP:

```
dcat:Catalog
  rdf:type sh:NodeShape ;
  sh:name "Catalog"@en ;
  sh:property [
    sh:path dct:description ;
    sh:minCount 1 ;
    sh:nodeKind sh:Literal ;
    sh:severity sh:Violation ;
  ] .
```

Listing 5. example of a DCAT-AP shape

The shape in Listing 5 doesn't use the sh:targetClass (https://www.w3.org/TR/shacl/#targetClass) property, while OSLO2 SHACL shapes use it, which means that subclasses (of the dcat:Catalog in the case) cannot inherit the shapes applied to it and benefit from the validation.

Third, by having the properties described as blank nodes, third party models which extend the original data models do not have the possibility to forbid (by means of sh:deactivated (https://www.w3.org/TR/shacl/#deactivated)) some shapes if they do not apply in their domain (imagine that a XYZ Public Service might have 2 descriptions while the CPSV-AP Public Service allows max 1 description), thus generating more violations than necessary.

Last, but not least, by explicitly naming SHACL properties, validation libraries which return SHACL validation reports might use the name of the shape by means of sh:sourceShape (https://www.w3.org/TR/shacl/#results-source-shape) that would be otherwise not accessible as they would be blank nodes.

There are two side effects to this way to describe SHACL shapes to be taken in account:

1. URIs for shapes needs to be defined;
2. As shapes are reused and inherited, there is much more need to evaluate conflicting shapes.

For the first effect, naming conventions could be applied; we have seen above an example "hasMin1Max1Shape-description" which explicit the cardinality and the name of the property but the name could be even opaque as "PS123".

In the second case, care has to be taken when there are several data-models in cascade owned by different organizations which define their own SHACL shapes; of course it might happen that some of them conflict (properties could have different cardinalities or types). In this case, SPARQL itself could be helpful to identify conflicting shapes among each other:

```
PREFIX sh: <http://www.w3.org/ns/shacl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?shape1 ?shape2 (count(?targetClass2) as ?distance)
WHERE {
  ?shape1 sh:targetClass ?targetClass1 .
  ?shape2 sh:targetClass ?targetClass3 .
  ?targetClass1 rdfs:subClassOf* ?targetClass2 .
  ?targetClass2 rdfs:subClassOf+ ?targetClass3 .

  ?shape1 sh:property ?p1 .
  ?p1 sh:path ?path1 .

  ?shape2 sh:property ?p2 .
  ?p2 sh:path ?path2 .
  ?p1 sh:minCount ?min1 .
  ?p2 sh:minCount ?min2 .
  FILTER (?path1 = ?path2) .
  FILTER (?min1 != ?min2) .
  FILTER (?targetClass1 != ?targetClass2)
}
GROUP BY ?shape1 ?shape2
ORDER BY DESC(?distance)
LIMIT 100
```

Listing 6. SPARQL query to find out SHACL shapes in conflict on the minimum cardinality

The SPARQL query in Listing 6 will take all shapes that are applied to classes which are interlinked via rdfs:subClassOf relations and check if the minimum cardinality (indicated by sh:minCount) are different, therefore checking a possible conflict; of course there could be other checks that could be done.

Applying inferencing and business rules before validating

One of the benefits of using linked data is that RDF provides the basis for vocabularies, such as RDFS and OWL, to define meaning for relations:

- RDFS (https://en.wikipedia.org/wiki/RDF_Schema), describes concepts like subclass and sub properties;
- OWL (https://en.wikipedia.org/wiki/Web_Ontology_Language), defines new type of relations, such as inverse, transitive and symmetric.

Thanks to a RDFS or an OWL reasoner (https://en.wikipedia.org/wiki/Semantic_reasoner), new relations or properties could be generated and added to the original data (e.g if the "friendship" relation is defined as transitive and A is friend of B and B is friend of C, then A is friend of C, thus it would be possible to create a more interconnected graph).

In addition, there have been different languages such as SPIN or SWRL that allow defining business rules able to enrich the current data by creating new relations or properties. More recently, SHACL allows to generate new relations by means of SHACL rules (https://www.w3.org/TR/shacl-af/#rules).

In both cases, adding new relations or properties might cause violation of the original data model, thus the need to perform validation after having enriched the data.

However, as the enrichment might be an expensive operation in terms of time, it should be an optional task for a validator to perform.

Having a flexible user interface

When performing a validation of RDF, one has to think who the end users are and how they might use such validation.

A web interface is definitely good for end users, however, it requires manual validation (by means file upload, etc.) and, considering that RDF should be at first managed by machines, such validation mechanism should be automated as much as possible. Thus the need to create a validation mechanism decoupled from the web interface and to which a web frontend could connect to.

SHACL validation can also run via a command line interface which could be integrated in a script to be run automatically. For example, TopQuadrant released a SHACL library and a command line interface where the user can specify the data file and the SHACL file to be checked against (https://github.com/TopQuadrant/shacl#command-line-usage).

A web interface is desirable to manage the validation process which in turn could trigger a data manipulation process.

A possible linked data validation pipeline

Having discussed the above points, I found out in Apache Nifi, an open source software framework, a valid data flow tool that could be easily put in place by people not having a technical background as:

- It has a web interface, therefore accessible through a web browser;
- It provides already many components (processors) that can be arbitrary connected between each other such as reading files in a folder, reading a file from URL or provide a web service waiting for data to be sent over;
- It provides data lineage over the pipeline so the end user can see at any steps if data is valid and how it is enriched;
- It allows to export the configuration of the data flow to be reused and imported by other users

Apache Nifi could be then used as basis for a linked data validation pipeline that could be connected to a web frontend for user input and reporting activities.

For the purpose of this paper, five types of validator processor have been created: SPARQL, SHACL, RDFS, OWL, and Custom Rules. Such processors are independent and can be put in any sequence the user needs, so the validation can run through different stages. If the data is valid for one of those processors, the same data is passed to the next processor; if data is not valid, an error is generated and sent to a dedicated processor that handles errors.

Further, each validator processor can connect to existing processors providing different way to the users to connect to.

The SPARQL validator processor could be run to validate user data but also to run the query previously provided to check if the SHACL shapes are in conflict between each other, thus the data modeler could use it.

The SHACL validator processor can make use of the SHACL shapes and validate thanks to the open source TopQuadrant SHACL API (https://github.com/TopQuadrant/shacl).

The RDFS, OWL and Custom Rules validator processors are based on Jena reasoners (https://jena.apache.org/documentation/inference/) that enrich the data and check that there are no inconsistencies.

In the image below a possible data flow is designed where, at any step, if the validation does not pass a validator processor, highlighted in green, the output is written to file saved to a folder.

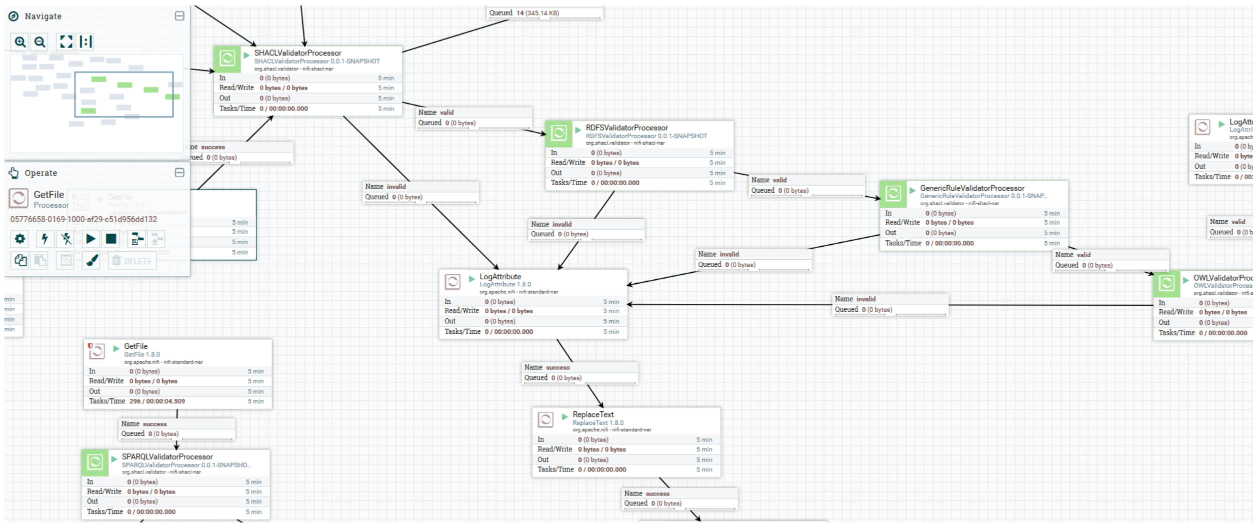

Figure 1. data flow in Apache Nifi (enlarge)

The source code of the validators has been released on GitHub (https://github.com/EmidioStani/shacl-processor) which can be compiled and reused directly in Apache Nifi.

Conclusions and future directions

Within this paper, the reader understands what are the implications of writing SHACL shapes and the limitations of the current approaches. A possible solution to create linked data validation pipeline is provided.

Developers could integrate this pipeline within their frontend on one hand and to storage facilities such as triple stores once the data passed the validation, to a reporting mechanism or to trigger a data manipulation process.

Note

[1] Estimates differ, but experts think organizations spend between 10-30% of revenue handling data quality issues. IBM estimated the cost of poor quality data in the US in 2016 was $3.1 trillions. – Source The DAMA Guide to the Data Management Body of Knowledge (DAMA-DMBOK) (https://technicspub.com/dmbok/)

About the author

Emidio is a technical expert in the Technology Consulting in Belgium. He is active in the fields of data management, semantic and technical interoperability as well as software engineering. He supports clients from designing data architecture, assist in the creation of data models and ensure data quality during ETL processes. You can reach Emidio via emidiostani@gmail.com or https://twitter.com/emidiostani.

Subscribe to comments: For this article | For all articles