Reusable SHACL Constraint Components for Validating Geospatial Linked Data

Christophe Debruyne, Kris McGlinn[2]

[1] Smals Research, Avenue Fonsny 20, 1060 Brussels, Belgium
christophe.debruyne@smals.be
[2] ADAPT Centre, Trinity College Dublin, Dublin 2, Ireland
kris.mcglinn@adaptcentre.ie

Abstract. SHACL provides us a powerful way of declaring validation rules for datasets. The built-in functions are quite limited, but we can use SPARQL to create custom constraint components. The problem is one could end up reinventing the wheel for constraints that hold in many contexts, such as topological relationships. We present GeoSHACL, a set of GeoSPARQL-based SHACL constraint components published as Linked Data. We thus provide constraint components that can be shared and reused. By starting with the topological relations of simple features, our goal is to provide a reusable set of such constraints. This article elaborates on some of the technical design decisions and provides a brief demonstration.

Keywords: Data Quality, Data Validation, SHACL, Geospatial Linked Data

1 Introduction

The Shapes Constraint Language (SHACL) [1] is a W3C Recommendation for validating RDF [2] graphs.[2] While it is oftentimes mentioned to validate Linked Data, the reality is a bit more nuanced; SHACL can validate RDF graphs in general. SHACL provides a set of "core" constructs for declaring rules (value- and data type checking, cardinality, value ranges, comparisons,… which can be combined with a set of logical operators). While those core constructs are arguably "limited" for modeling domain-specific constraints, SHACL does allow one to create custom components. One uses the SHACL vocabulary to declare new constraint components, but their "implementation" is done with SPARQL [3].[3]

The problem, however, is that many constraints may be "generic"; constraints that are general enough to be applicable in many domains. It is thus more than likely that different domain experts end up reinventing the wheel when creating constraints. For

[2] http://www.w3.org/ns/shacl# (with the usual namespace prefix sh)

[3] SHACL Advanced Features also specified a JavaScript extension for implementing rules and constraints in that language. This is, however, not important for this paper.

instance, most (Linked Data) datasets have some geospatial dimension [4]. That geospatial dimension is often a convenient way for aligning, integrating, and relating information on the Web. However, we may want to check whether certain topological constraints hold when doing so: Belgium should border France and countries should not overlap, for instance. These simple topological constraints are arguably commonplace and should be checked to ensure a geospatial dataset's quality. SHACL, however, does not provide support for topological relationships. To avoid the implementation of such constraint components over and over again, we propose **GeoSHACL**.

GeoSHACL is a set of constraint components that have been published according to best practices as a Linked Data vocabulary on the Web. The contributions of this paper are the dataset and its demonstration. The dataset can easily be retrieved and used by others as part of their validation processes. Thus, we also use this paper to advocate (a repository) of interoperable SHACL constraint components. W While one currently has to include the dataset in their own shapes graphs, it is hoped that the community will consider providing support for "importing" constraints, either via extensions of the SHACL standard or via tooling.

In this paper, we first introduce GeoSPARQL, a standard for representing and querying geospatial information on the Linked Data Web, as it provides the foundation for our topological relationships. Then we present GeoSHACL. We will focus on some of the design considerations and implementation details. GeoSHACL will be demonstrated with a simple example. We end the paper with some concluding remarks and some of the next steps that could be undertaken after this study.

2 GeoSPARQL

The OGC GeoSPARQL [5] standard proposes two things. First, it provides a vocabulary to represent geographical features and geometries, of which the latter can be expressed in either Well-Know Text (WKT) format or Geography Markup Language (GML). Features represent the "things" with a spatial dimension, such as a building, and geometries represent the spatial dimension (point, boundary, etc.). Features can be related with predicates such as `geo:hasGeometry`, or specializations thereof. Secondly, as its name implies, it specifies an extension to SPARQL for formulating geospatial queries. That extension consists mainly of two things:

1. Functions that yield a value. Examples include `geof:sfDisjoint` and `geof:sfIntersects` to determine whether the *lexical* representations of two geometries are, respectively, disjoint or intersecting.
2. A set of query-transformation rules to facilitate queries. For instance, a query with the triple pattern `?a geo:sfDisjoint ?b` looking for two features or geometries that are disjoint will be rewritten as a SPARQL query with UNION keywords to match five alternatives: one for looking for `?a geo:sfDisjoint ?b` as an asserted triple in the graph and four for all possible combinations of ?a and ?b being either bound to a feature or a geometry. These four alternatives then avail of the `geof:sfDisjoint` function.

Notice that there is a difference between **geo:**sfDisjoint and **geof:**sfDisjoint. While they have the same node-ID, both are declared in a different schema. The former is declared in the namespace of the GeoSPARQL ontology[4] and refers to the predicate. The latter is declared in GeoSPARQL's functions namespace[5] and refers to the functions that can be used in, for instance, filters.

It is important to note that while those transformation rules are part of the specification, not all implementations support those (by default). Some implementations require one to enable those rules explicitly.

3 GeoSHACL

GeoSPARQL is the standard for representing (complex) geospatial data on the Linked Data Web. There are some other (simple) standards for representing points in a coordinate system (e.g., longitude and latitude). We focus on GeoSPARQL for this study as these points can be converted into GeoSPARQL coordinates and GeoSPARQL supports more complex geometries. To demonstrate the viability of our approach, we first decided to focus on the so-called "simple feature relation family", which are the topological relations (and corresponding functions) that a GeoSPARQL-compliant system should support. These relations and functions are sfEquals, sfDisjoint, sfIntersects, sfTouches, sfCrosses, sfWithin, sfContains, and sfOverlaps. GeoSHACL must provide support for these eight relations.

One counterintuitive quirk of GeoSPARQL is that points can never be equal, even when you compare a point with itself. This is because a criterion for equality is that boundaries must be non-empty and shared and that points have, by definition, empty boundaries. Therefore, we have provided support for an "intuitive equals", which is based on two other relations (sfContains and sfWithin).

Two design decisions informed the development of GeoSHACL:

- We will not assume that transformation rules have been enabled, which means that the use of these constraints will rely on the lexical representations of geometries.
- A user should be able to compare the lexical representation of a geometry (via a path) with either a constant or the lexical representation of another geometry via a predicate. The behavior thus resembles those of SHACL core comparison operators.

We present below the specific implementation of one of the relations. All eight relations follow a similar pattern. The "intuitive equals" also uses the same pattern but uses two functions. Rather than "reusing" the predicates from GeoSPARQL, we declared predicates for the constraints in our GeoSHACL namespace. This is to avoid any

[4] http://www.opengis.net/ont/geosparql# (with the usual namespace prefix geo)
[5] http://www.opengis.net/def/function/geosparql/ (with the usual namespace prefix geof)

ambiguity when one would provide the shapes graph to a GeoSPARQL-enabled triple-store. On line 14, we test the case a constant was provided for $touches. If no lexical representation (e.g., a WKT or GML literal) is bound to either $touches or $value, this one fails. Lines 16-18 consider the case a predicate was provided by the user. In that case, $touches should contain an IRI (line 16) used as the predicate of a triple pattern (line 17). The value via that predicate is then passed along with the value of $value to the GeoSPARQL function (line 18). We finally note that the classes sh:ConstraintComponent and sh:SPARQLAskValidator are, of course, declared in the SHACL vocabulary.

```
1.  # Implementation of geof:sfTouches constraints
2.  geosh:touchesConstraint
3.    a sh:ConstraintComponent ;
4.    sh:parameter [
5.      sh:path geosh:touches ;
6.    ] ;
7.    sh:validator [
8.      a sh:SPARQLAskValidator ;
9.      sh:message "Value does not touch {$touches}." ;
10.     sh:ask """
11.       PREFIX geo: <http://www.opengis.net/ont/geosparql#>
12.       PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
13.       ASK {
14.         { FILTER( geof:sfTouches($value, $touches) ) }
15.         UNION {
16.           FILTER( isIRI($touches) )
17.           $this $touches ?otherValue .
18.           FILTER( geof:sfTouches($value, ?otherValue) )
19.         }
20.       }""" ;
21.   ] ;
22. .
```

One could argue that the function could have been provided as an argument for the constraint component, thereby reducing the number of components. SHACL does not provide support for using variables to place function calls, however. In other words, SHACL does not support the use of a variable where functions calls are made, as variables must contain RDF terms. Another approach could have been to test for the different values in one large query. Not only would that have impeded the efficiency of the approach (i.e., computational overhead), it would have made our approach less extensible. The inclusion of a new relation merely requires extending the shapes graph and not changing the query.

While not an ontology in the traditional sense (e.g., an OWL 2 ontology), we have stored GeoSHACL as an instance of an ontology[6]. This allowed us to provide metadata for both the ontology and the constraint components we have developed. The documentation of the ontology was generated with WIDOCO [6]. The artifact has been published according to best practices and guidelines within the community (e.g., the use of permanent URIs, content negotiation, and a permissible license). GeoSHACL contains the implementation of nine constraint components. These correspond with the eight simple

[6] Namespace geosh: https://w3id.org/geoshacl#

relations and our "intuitive equals".

4 Demonstration

In this section we demonstrate GeoSHACL. This simple example will illustrate the use of the intuitive equals and the use of both a constant and a predicate in our shapes. Our data graph looks as follows (prefixes are omitted):

```
ex:Point1 a geo:Feature, ex:Point ;
  geo:hasGeometry [
    a geo:Geometry ;
    geo:asWKT "Point(1 1)"^^geo:wktLiteral
  ] ;
.
ex:Point2 a geo:Feature, ex:Point ;
  geo:hasGeometry [
    a geo:Geometry ;
    geo:asWKT "Point(2 2)"^^geo:wktLiteral ;
  ] ;
.
ex:SquareGeom a geo:Geometry ;
  geo:sfContains ex:Point2, ex:PPoint1 ;
  geo:asWKT "POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))"^^geo:wktLiteral ;
.
```

In our data graph, we have two features (points), each with a geometry, and another geometry representing a square. The square asserts that it contains both points, though one can see that one of the points lies outside the square. Our shapes graph using GeoSHACL looks as follows (prefixes are omitted):

```
# Check whether all points are equal with Point(1 1)
ex:PointShape
  a sh:NodeShape ;
  sh:targetClass ex:Point ;
  sh:property [
    sh:path (geo:hasGeometry geo:asWKT) ;
    geosh:intuitiveEquals "Point(1 1)"^^geo:wktLiteral
  ] ;
.
# Are things that geometries contain actually within?
ex:SquareGeomShape
  a sh:NodeShape ;
  sh:targetClass geo:Geometry ;
    sh:property [
    sh:path (geo:sfContains geo:hasGeometry geo:asWKT) ;
    geosh:within geo:asWKT ;
  ] ;
.
```

After passing both the data graph and the shapes graph to a SHACL engine, which for our demo is the Apache Jena implementation[7], the engine can detect the errors using

[7] https://jena.apache.org/documentation/shacl/

the GeoSPARQL functions (see Listing 1).

```
Node=<http://www.example.org/Point2>
  Path=<http://...#hasGeometry>/<http://...#asWKT>
  Value: "Point(2 2)"^^<http://...#wktLiteral>
  Message: Value is not intuitively equal to Point(1 1)
Node=<http://www.example.org/SquareGeom>
  Path=(<http://...#sfContains>/<http://...#hasGeometry>)/<http://...#asWKT>
  Value: "Point(2 2)"^^<http://...#wktLiteral>
  Message: Value is not within <http://...#asWKT>.
```

Listing 1. The validation report after validating the data graph with the shapes graph

Even though our example uses WKT to represent geometries, this does not mean that our solution is solely intended for WKT. Apache Jena's GeoSPARQL implementation also supports GML. For GeoSHACL to work, the underlying SPARQL engine needs to (correctly) support GeoSPARQL. If that is not the case, the user may not notice that the validation process fails, and that the validation report may be invalid. For example, when GeoSPARQL functions are not supported, one will observe that those FILTERs will "fail gracefully" as the error within the FILTER results in a solution not being withheld. When using GeoSHACL, one has to ensure that the GeoSPARQL engine is compliant by using benchmarks such as [7]. It is possible to define SHACL rules that test the existence of GeoSPARQL functions, however.

5 Discussion

Due to the flexible nature of data represented using Semantic Web technologies, SHACL constraints have been proposed as a solution for validating geospatial datasets by the W3C working group who developed the Spatial Data on the Web Best Practices" [8]. In practice, there are few examples of their application. In [9], Huang et al. present the use of SHACL for validating the results of the integration of geospatial and traffic data to ensure that semantic correctness is maintained at different levels of detail within the representations of geospatial data. In [10], Stolk et al. demonstrate the use of SHACL constraints for validating a Building Information Model standard called Industry Foundations Classes, which itself has been derived from a GIS geometry.

In this study, we considered the support for the eight topological relations that are part of the simple features specifications. While seemingly limited, we consider this an important step toward shareable and reusable SHACL constraints. Given the growing interest in representing geospatial data as Linked Data, this work will play an important role in providing a set of reusable SHACL constraints for researchers who wish to validate their data represented using GeoSPARQL. We foresee the support of other constraint components and consider validating the relationships between features and geometries (next to referring to literals) as logical next steps.

6 Conclusions

We presented GeoSHACL, which provides shareable and reusable constraint components for topological relations between simple features. GeoSHACL is built on top of

GeoSPARQL. The motivation of this work is that while SHACL is powerful, one also has to consider sharing and reusing constraint components that may hold in many domains, applications, etc. While seemingly simple, we hope that this paper provides a first step towards realizing this.

With respect to GeoSHACL, there is room for future work. One is the inclusion of the other topological relations that GeoSPARQL provides. As we do not assume that SPARQL engines support the transformation rules, we aim to investigate how we could rewrite or extend the queries in GeoSHACL to refer to features and geometries next to the lexical representations. And while the SPARQL engine's support for (and compliance with) GeoSPARQL falls outside the scope of GeoSHACL, GeoSHACL can be extended to test the availability of GeoSPARQL functions.

Acknowledgements. Kris McGlinn is supported by the ADAPT Centre for Digital Content Technology, which is funded under the SFI Research Centres Programme (Grant 13/RC/2106) and is co-funded under the European Regional Development Fund.

References

1. Knublauch, H., Kontokostas, D.: Shapes Constraint Language (SHACL), https://www.w3.org/TR/shacl/.
2. Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1 Concepts and Abstract Syntax, https://www.w3.org/TR/rdf11-concepts/.
3. Seaborne, A., Harris, S.: SPARQL 1.1 Query Language, https://www.w3.org/TR/sparql11-query/.
4. Shadbolt, N., O'Hara, K., Berners-Lee, T., Gibbins, N., Glaser, H., Hall, W., Schraefel, M.C.: Linked Open Government Data: Lessons from Data.gov.uk. IEEE Intell. Syst. 27, 16–24 (2012). https://doi.org/10.1109/MIS.2012.23.
5. Open Geospatial Consortium: OGC GeoSPARQL - A Geographic Query Language for RDF Data, https://www.ogc.org/standards/geosparql.
6. Garijo, D.: WIDOCO: A wizard for documenting ontologies. In: The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part II. pp. 94–102. Springer (2017). https://doi.org/10.1007/978-3-319-68204-4_9.
7. Jovanovik, M., Homburg, T., Spasić, M.: Software for the GeoSPARQL compliance benchmark. Softw. Impacts. 8, 100071 (2021). https://doi.org/10.1016/j.simpa.2021.100071.
8. Tandy, J., van den Brink, L., Barnaghi, P.: Spatial Data on the Web Best Practices, https://www.w3.org/TR/sdw-bp/.
9. Huang, W., Kazemzadeh, K., Mansourian, A., Harrie, L.: Towards Knowledge-Based Geospatial Data Integration and Visualization: A Case of Visualizing Urban Bicycling Suitability. IEEE Access. 8, 85473–85489 (2020). https://doi.org/10.1109/ACCESS.2020.2992023.
10. Stolk, S., McGlinn, K.: Validation of IfcOWL datasets using SHACL. In: Proceedings of the 8th Linked Data in Architecture and Construction Workshop Dublin, Ireland, June 17-19, 2020 (virtually hosted). pp. 91–104 (2020).