

Shape Constraint Language

Veronika Heimsbakk
veronika.heimsbakk@acando.no



24. april 2017

Agenda

1. Terminology
2. What is SHACL?
3. Using SHACL for data validation
4. Validation result graphs
5. SPARQL processing
6. Demo

RDF graph An **RDF graph** is a set of *triples*.

RDF triple An **RDF triple** consists of three components; *subject*, *predicate* and *object*.

RDF graph An **RDF graph** is a set of *triples*.

RDF triple An **RDF triple** consists of three components; *subject*, *predicate* and *object*.

subject is an *IRI* or a *blank node*.

RDF graph An **RDF graph** is a set of *triples*.

RDF triple An **RDF triple** consists of three components; *subject*, *predicate* and *object*.

subject is an *IRI* or a *blank node*.

predicate is an *IRI*.

RDF graph An **RDF graph** is a set of *triples*.

RDF triple An **RDF triple** consists of three components; *subject*, *predicate* and *object*.

subject is an *IRI* or a *blank node*.

predicate is an *IRI*.

object is an *IRI*, *literal* or a *blank node*.

IRI (*Internationalized Resource Identifier*) within an RDF graph is a Unicode string with syntax defined in RFC3987. IRIs must be absolute, and may contain fragment identifier.

Shapes Constraint Language

A language for describing and validating RDF graphs.

- ▶ A vocabulary defined by a W3C working group.
- ▶ First revision in July 2015.
- ▶ Became a Candidate Recommendation 11th of April 2017.

sh: <https://www.w3.org/ns/shacl#>

The syntax of SHACL is the same as for any other RDF-based standard.

Today we will see both **Turtle** and **JSON-LD**.

Shapes

Shape

Definition

A **shape** is an IRI or blank node **s** that fulfills at least one of the following conditions in the shapes graph:

- ▶ **s** is a SHACL instance of **sh:NodeShape** or **sh:PropertyShape**.
- ▶ **s** is subject of a triple that has **sh:targetClass**, **sh:targetNode**, **sh:targetObjectsOf** or **sh:targetSubjectsOf** as predicate.
- ▶ **s** is subject of a triple that has a parameter as predicate.
- ▶ **s** is a value of a shape-expecting, non-list-taking parameter such as **sh:node**, or a member of a SHACL list that is a value of a shape-expecting and list-taking parameter such as **sh:or**.

Node shape

Definition

A **node shape** is a shape in the shapes graph that is not the subject of a triple with **sh:path** as its predicate. SHACL instances of **sh:NodeShape** cannot have a value for the property **sh:path**.

Property shape

Definition

A property shape is a shape in the shapes graph that is the subject of a triple that has `sh:path` as its predicate. A shape has at most one value for `sh:path`. Each value of `sh:path` in a shape must be a well-formed SHACL property path. SHACL instances of `sh:PropertyShape` have one value for the property `sh:path`.

It is recommended, but not required, for a node or property shape to be declared as a SHACL instance of `sh:NodeShape` or `sh:PropertyShape`.

Building a shape

```
ex:PersonShape
  a sh:NodeShape ;
  sh:targetClass ex:Person .
```

Building a shape

```
ex:NamePropertyShape
  a sh:PropertyShape ;
  sh:path ex:name ;
  sh:minCount 1 .
```

Building a shape

```
ex:PersonShape
  a sh:NodeShape ;
  sh:targetClass ex:Person ;
  sh:property [
    sh:path ex:name ;
    sh:minCount 1 ;
  ] .
```

Severity

sh:Info

A non-critical informative message.

sh:Warning

A non-critical constraint violation indicating a warning.

sh:Violation

A constraint violation.

Building a shape

```
ex:PersonShape
  a sh:NodeShape ;
  sh:targetClass ex:Person ;
  sh:property [
    sh:path ex:name ;
    sh:minCount 1 ;
    sh:severity sh:Violation ;
  ] .
```

Focus node An RDF term that is validated against a shape using the triples from a data graph is called a **focus node**.

Value nodes The validators of most constraint components use the concept of value nodes, which is defined as follows:

- ▶ For node shapes the value nodes are the individual focus nodes, forming a set with exactly one member.
- ▶ For property shapes with a value for `sh:path` p the value nodes are the set of nodes in the data graph that can be reached from the focus node with the path mapping of p .

SHACL Core constraint components

- ▶ Value type
- ▶ Cardinality
- ▶ String-based
- ▶ Property pair
- ▶ Logical
- ▶ Shape-based
- ▶ Other

Value Type Constraint Components

Used to restrict the type of value nodes.

- sh:class** Type of all value nodes. Values are IRIs.
- sh:datatype** Datatype of all value nodes. Values are IRIs. At most one value for sh:datatype.
- sh:nodeKind** Node kind of all value nodes. E.g. sh:Literal.

Value Type Constraint Components

```
ex:PersonShape
  a sh:NodeShape ;
  sh:property [
    sh:path ex:name ;
    sh:datatype xsd:string ;
  ] .
```

Cardinality Constraint Components

Restriction on the number of value nodes for the given focus node.

sh:minCount Minimum cardinality.

sh:maxCount Maximum cardinality.

Only for property shapes. At most one value, literal integer.

Cardinality Constraint Components

```
ex:PersonShape
  a sh:NodeShape ;
  sh:property [
    sh:path ex:name ;
    sh:minCount 1 ;
    sh:maxCount 1 ;
  ] .
```

Value Range Constraint Components

`sh:minExclusive` <
`sh:minInclusive` <=
`sh:maxExclusive` >
`sh:maxInclusive` >=

String-based Constraint Components

Specify conditions on the string representation of value nodes.

sh:pattern A regex that all value nodes need to match. String literals, valid arguments for SPARQL REGEX function.

sh:languageIn SHACL list of allowed language tags for each value node. List members are string literals.

sh:minLength, **sh:maxLength**, **sh:uniqueLang**

String-based Constraint Components

```
ex:PersonShape
  a sh:NodeShape ;
  sh:property [
    sh:path ex:name ;
    sh:pattern "^V" ;
  ] .
```

Property Pair Constraint Components

Specify conditions on the sets of value nodes in relation to other properties.

- ▶ `sh:equals`
- ▶ `sh:disjoint`
- ▶ `sh:lessThan`
- ▶ `sh:lessThanOrEquals`

Logical Constraint Components

Implement the common operators and, or, not and a variation of exclusive or.

- ▶ `sh:not`
- ▶ `sh:and`
- ▶ `sh:or`
- ▶ `sh:xone`

Shape Based Constraint Components

Can be used to specify complex conditions by validating the value nodes against certain shapes.

- ▶ `sh:node`
- ▶ `sh:property`
- ▶ `sh:qualifiedValueShape`
- ▶ `sh:qualifiedMinCount`
- ▶ `sh:qualifiedMaxCount`

Other Constraint Components

- ▶ `sh:closed`
- ▶ `sh:ignoredProperties`
- ▶ `sh:hasValue`
- ▶ `sh:in`

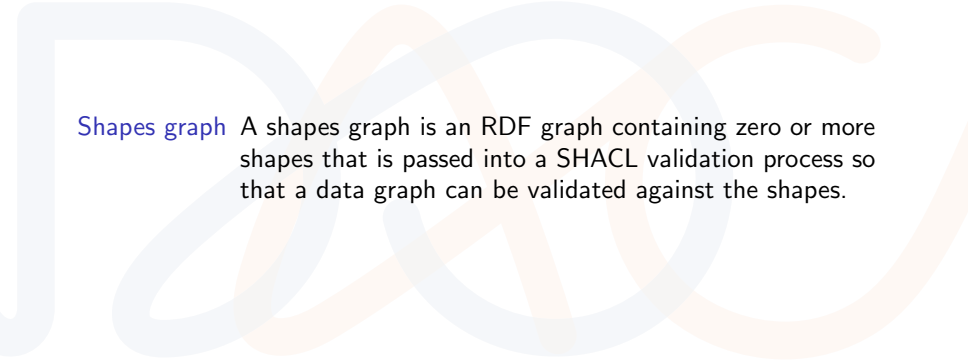
A node shape containing several property shapes

```
ex:PersonShape
  a sh:NodeShape ;
  sh:property [
    sh:path ex:name ;
    sh:datatype xsd:string ;
    sh:minCount 1 ;
  ] ;
  sh:property [
    sh:path ex:age ;
    sh:datatype xsd:integer ;
  ] .
```

After the lecture, I got a question about several property shapes. Here is an example of a node shape with two property shapes. Note that **sh:path** only may be declared *once* per property shape.

Validation Result Graphs

Validation takes a **data graph** and a **shapes graph** as input and produces a validation report containing the results of the validation.



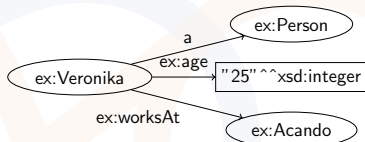
Shapes graph A shapes graph is an RDF graph containing zero or more shapes that is passed into a SHACL validation process so that a data graph can be validated against the shapes.

Data graph Any RDF graph can be a data graph.

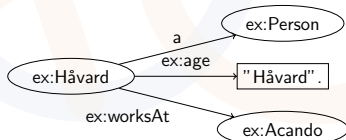
Validation report The validation report is the result of the validation process that reports the conformance and the set of all validation results. The validation report is described with the SHACL Validation Report Vocabulary .

Example data graph

```
ex:Veronika
  a ex:Person ;
  ex:age "25"^^xsd:integer ;
  ex:worksAt ex:Acando .
```

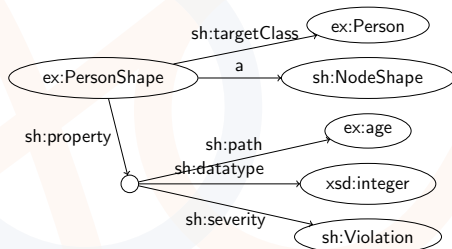


```
ex:Håvard
  a ex:Person ;
  ex:age "Håvard" ;
  ex:worksAt ex:Acando .
```



Example shapes graph

```
ex:PersonShape
  a sh:NodeShape ;
  sh:targetClass ex:Person ;
  sh:property [
    sh:path ex:age ;
    sh:datatype xsd:integer ;
    sh:severity sh:Violation ;
  ] .
```



Validation result graph

```
[
  a sh:ValidationReport ;
  sh:conforms false ;
  sh:result [
    a sh:ValidationResult ;
    sh:resultSeverity sh:Violation ;
    sh:focusNode ex:Håvard ;
    sh:resultPath ex:age ;
    sh:resultMessage "ex:age is expecting xsd:integer" ;
    sh:sourceConstraintComponent sh:DatatypeConstraintComponent ;
    sh:sourceShape ex:PersonShape ;
  ]
] .
```


Summary

- ▶ Introducing constraints is important for data consistency, validation, quality and for making sense.
- ▶ Expressing constraints (SHACL) \neq formal statements in predicate logic (OWL).
- ▶ May achieve SPARQL free validation.

More on SHACL

- ▶ <https://www.linkedin.com/pulse/breaking-news-last-shacl-made-cr-status-jan-voskuil>
- ▶ <http://www.topquadrant.com/2015/07/29/shacl-the-next-generation-data-modeling-language/>
- ▶ <https://www.w3.org/TR/shacl/>

Next up: Håvard & SPARQL processing