# USING JENA AND SHACL TO VALIDATE RDF DATA

HENRIETTE HARMSE

RDF enables users to capture data in a way that is intuitive to them. This means that data is often captured without conforming to any schema. It is often useful to know that an RDF dataset conforms to some (potential partial) schema. This is where SHACL (**SHA**pe **C**onstraint **L**anguage), a W3C standard, comes into play. It is a language for descibing and validating RDF graphs [1]. In this post I will give a brief use SHACL to validate RDF data using the Jena implementation of SHACL.

## 1. A SHACL Example

We will use an example from the SHACL specification. Assume we have the a file `person.ttl` that contains the following data:

```
@prefix ex: <http://example.com/ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ex:Alice
        a ex:Person ;
        ex:ssn "987-65-432A" .

ex:Bob
        a ex:Person ;
        ex:ssn "123-45-6789" ;
        ex:ssn "124-35-6789" .

ex:Calvin
        a ex:Person ;
        ex:birthDate "1971-07-07"^^xsd:date ;
        ex:worksFor ex:UntypedCompany .
```

To validate this data we create a shape definition in `personShape.ttl` containing:

```
@prefix dash: <http://datashapes.org/dash#> .
@prefix ex: <http://example.com/ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
```

*Date*: 11th March 2018.

```
ex:PersonShape
        a sh:NodeShape ;
        sh:targetClass ex:Person ;    # Applies to all persons
        sh:property [                 # _:b1
                sh:path ex:ssn ;           # constrains the values of ex:ssn
                sh:maxCount 1 ;
                sh:datatype xsd:string ;
                sh:pattern "^\\d{3}-\\d{2}-\\d{4}$" ;
        ] ;
        sh:property [                 # _:b2
                sh:path ex:worksFor ;
                sh:class ex:Company ;
                sh:nodeKind sh:IRI ;
        ] ;
        sh:closed true ;
        sh:ignoredProperties ( rdf:type ) .
```

## 2. A Code Example using Jena

To validate our RDF data using our SHACL shape we will use the Jena implementation of SHACL. Start by adding the SHACL dependency to your Maven `pom.xml`. Note that you do not need to add Jena as well as the SHACL pom already includes Jena.

```
<dependency>
  <groupId>org.topbraid</groupId>
  <artifactId>shacl</artifactId>
  <version>1.0.1</version>
</dependency>
```

In the code we will assume the `person.ttl` and `personShape.ttl` files are in `$Project/src/main/resources/`. The code for doing the validation is the following then:

```
package org.shacl.tutorial;

import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStream;
import java.nio.file.Path;
import java.nio.file.Paths;

import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.Resource;
import org.apache.jena.riot.RDFDataMgr;
import org.apache.jena.riot.RDFFormat;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.slf4j.Marker;
import org.slf4j.MarkerFactory;
import org.topbraid.shacl.validation.ValidationUtil;
import org.topbraid.shacl.vocabulary.SH;
import org.topbraid.spin.util.JenaUtil;
```

```
public class ShaclValidation {
  private static Logger logger = LoggerFactory.getLogger(ShaclValidation.class);
  // Why This Failure marker
  private static final Marker WTF_MARKER = MarkerFactory.getMarker("WTF");

  public static void main(String[] args) {
    try {
      Path path = Paths.get(".").toAbsolutePath().normalize();
      String data = "file:" + path.toFile().getAbsolutePath() +
          "/src/main/resources/person.ttl";
      String shape = "file:" + path.toFile().getAbsolutePath() +
          "/src/main/resources/personShape.ttl";

      Model dataModel = JenaUtil.createDefaultModel();
      dataModel.read(data);
      Model shapeModel = JenaUtil.createDefaultModel();
      shapeModel.read(shape);

      Resource reportResource =
          ValidationUtil.validateModel(dataModel, shapeModel, true);
      boolean conforms  = reportResource.getProperty(SH.conforms).getBoolean();
      logger.trace("Conforms = " + conforms);

      if (!conforms) {
        String report = path.toFile().getAbsolutePath() +
            "/src/main/resources/report.ttl";
        File reportFile = new File(report);
        reportFile.createNewFile();
        OutputStream reportOutputStream = new FileOutputStream(reportFile);

        RDFDataMgr.write(reportOutputStream, reportResource.getModel(),
            RDFFormat.TURTLE);
      }
    } catch (Throwable t) {
      logger.error(WTF_MARKER, t.getMessage(), t);
    }
  }
}
```

## 3. Running the Code

Running the code will cause a `report.ttl` file to be written out to `$Project/src/main/resources/`. We can determine that our data does not conform by checking the `sh:conforms` property. We have 4 violations of our `ex:PersonShape`:

1. For `ex:Alice` the `ex:ssn` property does not conform to the pattern defined in the shape.
2. `ex:Bob` has 2 `ex:ssn` properties.
3. `ex:Calvin` works for a company that is not of type `ex:Company`.
4. `ex:Calvin` has a property `ex:birthDate` that is not allowed by `ex:PersonShape` since it is close by `sh:closed true`.

A corrected version of our person data may look as follows:

```
@prefix ex: <http://example.com/ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ex:Alice
        a ex:Person ;
        ex:ssn "987-65-4321" .

ex:Bob
        a ex:Person ;
        ex:ssn "124-35-6789" .

ex:ACECompany a ex:Company .

ex:Calvin
        a ex:Person ;
        ex:worksFor ex:ACECompany .
```

## 4. Conclusion

In this post I given a brief overview of how SHACL can be used to validate RDF data using the SHACL implementation of Jena. This code example is available at shacl tutorial.

## References

1. H. Knublauch and D. Kontokostas, *Shapes Constraint Language (SHACL)*, url=https://www.w3.org/TR/shacl/, 2017.