RDF Shape Rule Language

SHRL – pronounced as shurl

Dave Raggett, 14 December 2016

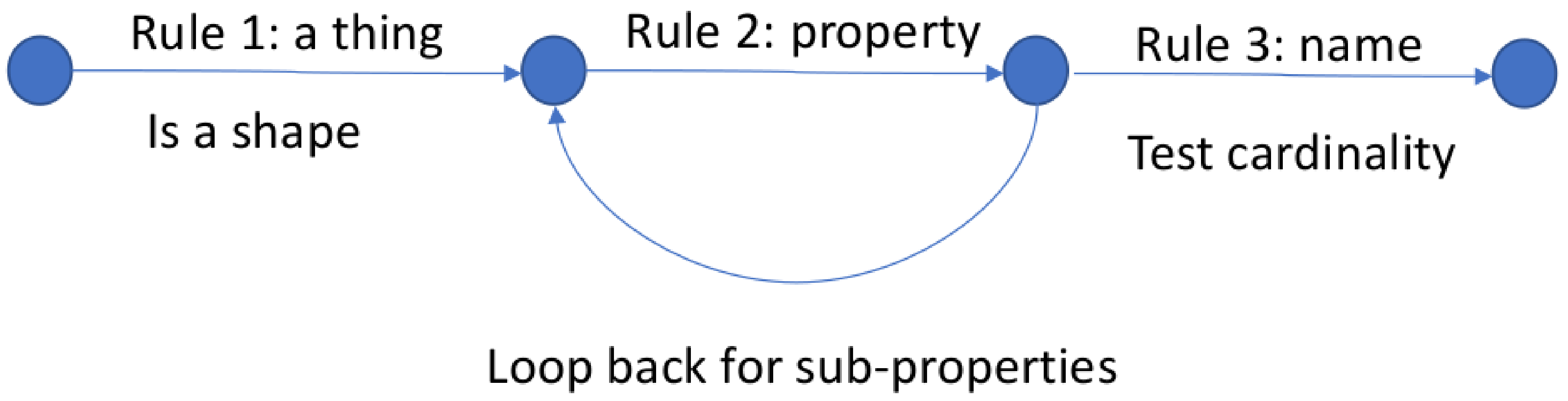
Open source implementation in JavaScript

Shape Rule Language (SHRL)

- The ability to validate data is important for data processing
- XML Schema can be used to validate data in XML
- We likewise need a validation language for RDF
  - A means to validate a collection of triples
    - Each triple is defined by its subject, predicate and object
    - RDF Schema and OWL focus on inferencing not validation
- SHRL is based upon augmented transition networks (ATNs)
  - ATNs were developed in early 1970's for natural language processing
  - ATNs can be readily applied to traversing RDF graphs
  - Simpler and easier to understand than alternatives e.g. SHACL and ShEx
  - Facilitates graphic views and editing of rules
- A rule graph defining a set of shapes is applied to a data graph

Shape Rule Language

- RDF implementation of Augmented Transition Networks



- A thing has zero or more properties, actions and events
- Properties may have sub properties
- Properties, actions and events must have one name

The shape in RDF using Turtle

```
@prefix sh: <http://www.w3.org/ns/shrl#>
@prefix td: <http://www.w3.org/ns/td#>
@prefix ex: <http://example.com/ns#>

ex:rule1
     a sh:Shape ;
     sh:targetClass td:thing ;
     sh:and ex:rule2 .
ex:rule2
     sh:rel td:property ;
     sh:and ex:rule3 , ex:rule2 .
ex:rule3
     rdfs:comment "Every property must have exactly one name" ;
     sh:rel td:name ;
     sh:minCount 1 ;
     sh:maxCount 1 .
```

n.b. the names spaces are yet to be standardizrd and are given for illustration purposes only

Explanation

- Rule 1 states that it defines a shape and matches RDF nodes that have rdf:type td:thing, i.e. nodes that represent a thing
- Rule 2 traverses from subject to object for triples with the predicate td:property
    - This rule loops back on itself to handle properties with sub-properties
- Rule 3 traverses triples with the predicate td:name and applies cardinality constraints to check that there is one and only one name
- If a rule's constraints are violated, its comment is used as an error message

SHRL Features

- Each shape must be indicated by a rule with **a sh:shape***
  - In other words, this rule is an instance of the class "shape"
  - Each such rule must select a set of nodes in the data graph
- You can select a particular data node with **sh:targetNode *node***
- Or select all data nodes that have a given class
  - **sh:targetClass *class*** – selects all x such that "x a class"
- Or select all subjects with a given predicate using **sh:rel *predicate***
  - Select all x such that "x predicate y"
- Or select all objects with a given predicate using **sh:rev *predicate***
  - Select all y such that "x predicate y"
- Or select nodes on the basis of the predicate class
  - **sh:relClass** and **sh:revClass** by analogy with targetClass, rel and rev above
- You may use sh:shape with more than one of the above selectors

*Turtle abbreviates “x rdf:type y” as “x a y”

Each rule traverses one or more triples

- Rules without sh:shape must have one of the following
- **sh:rel** – follow the given predicate from subject to object
- **sh:rev** – follow the given predicate from object to subject
- **sh:relClass** – follow predicates that are instances of the given class
  - Transitioning from subject to object
- **sh:revClass** – follow predicates that are instances of the given class
  - Transitioning from object to subject
- The traversal determines the set of data nodes to be passed to the rule's successors

SHRL Constraints

- **sh:minCount** and **sh:maxCount** – cardinality constraints on the triples traversed by the rule
- **sh:min** and **sh:max** – range constraints on numeric literal nodes
- **sh:value** – the node must have the given value
- **sh:match** – regular expression constraining string literal nodes
- **a sh:string** – the node must be a string literal
- **a sh:number** – the node must be a numeric literal
- **a sh:integer** – the node must be an integer literal
- **a sh:boolean** – the node must be a boolean literal
- a **sh:nonLiteral** – the node must not be a literal

Rules are chained together

- Rules are chained together with one of the following predicates that designate successor rules
- **sh:and** – all of the successors to this rule must be valid
- **sh:or** – at least one of the successors to this rule must be valid
- **sh:one** - exactly one of the successors to this rule must be valid
- **sh:not** – this rule is only valid if all of its successors are invalid
- If a successor rule is **a :sh:shape**, its selectors are ignored

Example rule

```
ex:rule2
   sh:rel td:property ;
   sh:and ex:rule3 , ex:rule 2 .
```

Augmentation with Application Code

- You can augment the transition network with application code for
  - Constraints across different properties of a thing
  - Constraints on the number of levels of nested properties
  - Constraints across metadata at different levels of a thing's object model
  - Generating output as a side effect of "recognizing a shape"
  - **sh:scope** – declares a new named scope
    - Inner scopes override outer scopes with the same name
  - **sh:eval** – invoke the named application defined function
    - Operates on the current data node and scope chain
    - Is called after processing the successors for this rule
    - The function should return false to indicate a failed test
- Future work will allow rule actions to be defined in RDF itself