Diverging Views of SHACL

Peter F. Patel-Schneider pfpschneider@gmail.com

Nuance Commmunications

SHACL is a new recommendation being developed by the W3C Data Shapes Working Group. SHACL is designed to address the need for a declarative language to validate or describe the contents of an RDF graph. This amounts roughly to checking whether an RDF graph satisfies a set of constraints. However, there are several diverging views of just how RDF graph validation should work, what kinds of constraints are needed for it, and how they should interact. This led to some difficult discussions in the working group. I will discuss these diverging views and how the current definition of SHACL matches against them.

This talk assumes familiarity with RDF and RDFS and some knowledge of OWL and SPARQL.

Basis of this Talk

- Current documents on SHACL
  - *SHACL working draft [1] and its current editors' draft*
- My participation in the W3C Data Shapes Working Group
  - *I am no longer a member as Nuance has quit W3C*
- My analyses of and work on related validation formalisms
  - *Description Logics Constraints [4,5], Shape Expressions [2]*
- My analyses of current commercial validation systems for RDF
  - *Stardog ICV [7], SPIN Constraints [6]*
- My analyses of and work on SPARQL,
  - *particularly EXISTS [8]*

Shapes Constraint Language (SHACL)

- Designed to determine whether the data in an RDF graph is directly suitable for processing by some application
  - *Is there sufficient information in the graph (maybe after inferencing)?*
  - *Does the information in the graph match expectations?*
- SHACL *validates* the contents of an RDF graph against some expectations
- Secondary purposes
  - *Describe which graphs validate against some shapes*
  - *Help build user and other interfaces that build or use valid graphs*
- Product of W3C Data Shapes Working Group
  - *More information at www.w3.org/2014/data-shapes*
  - *Main document: Shapes Constraint Language (SHACL)*

Why Care About SHACL

- It's a potential W3C recommendation
- It fills a gap in the W3C Semantic Web - validating content of RDF graphs
- It could end up as a suceessor to SPIN Constraints [6] and maybe to Stardog ICV [7]

Divergent Views of Validating RDF Graphs

- Work on SHACL started with groups espousing two divergent views
    1. *Schema View (from Shape Expressions [2])*
        - It's like schema recognition (in, e.g., Relax NG), but unordered.
    2. *Constraint View*
        - It's just checking integrity constraints.
            1. OWL variant (from description logic constraints [4], Stardog ICV [7])
                - Constraints are OWL axioms intepreted in a closed world setting
            2. SPARQL variant (from SPIN Constraints [6])
                - Constraints are just pretty syntax for SPARQL queries.
                    1. Just SPARQL. 2. SPARQL plus extensions.
- This talk is about these divergent views and variants and how they relate to the current version of SHACL.

Semantic Web

RDF Graph

```
:Mary rdf:type :Taxpayer;
   :name "Mary Smith";  :age 37;
   :spouse :Chris;
   :dependent :Chris, :Jeff;
   :SSN 123456789 .
:Chris rdf:type :Person;
   :name "Chris Smith"; :age 36;
   :SSN 234567890 .
:Jeff rdf:type :Dependent;
   :name "Jeff Smith";  :age 9;
   :gender male .
:Susan rdf:type :Person;
   :name "Susan Smith" .
```

Either RDFS Ontology

```
:Taxpayer rdfs:subClassOf :Person .
:Dependent rdfs:subClassOf :Person .
:name rdfs:range xsd:string .
:age rdfs:range xsd:integer .
:spouse rdfs:domain :Person; rdfs:range :Person.
:dependent rdfs:domain :Person;
           rdfs:range :Dependent .
```

Or OWL Ontology

```
:Person ⊑ =1:name ∧ ∀:name xsd:string
   ∧ =1:age ∧ ∀:age xsd:integer
   ∧ ∀:spouse :Person
:Dependent ⊑ :Person ∧ =1:SSN ∧ ∀:SSN xsd:integer
:Taxpayer ⊑ :Person
   ∧ =1:SSN ∧ ∀:SSN xsd:integer
   ∧ ∀:dependent :Dependent
```

This all works fine, as far as it goes, but there is something missing.

What's Missing?

```
:Mary rdf:type :Taxpayer;
  :name "Mary Smith";   :age 37;
  :dependent :Chris, :Jeff;
  :spouse :Chris;
  :SSN 123456789 .
:Chris rdf:type :Person;
  :name "Chris Smith";  :age 36;
  :SSN 234567890 .
:Jeff rdf:type :Dependent;
  :name "Jeff Smith";   :age 9;
  :gender male .
:Susan rdf:type :Person;
  :name "Susan Smith" .
```

- Only typing for :Chris is :Person
- No :SSN provided for :Jeff
- Extraneous :gender for :Jeff
- :Susan disconnected

- Want to be able to discover these problems
  - *Application may barf if it doesn't see expected values or may require values, e.g., to fill out forms*
  - *Describe actual output or required input for application*
- Want to be able to "validate" an RDF graph against some expectations.

Validation

- But what is validation (for RDF graphs)?
- What is validation related to?
- How can validation be specified?
- Two different views of validation:
  - *Validation as constraint checking*
  - *Validation as schema recognition*

Constraint View of Validation

- Validation is just checking constraints (against the graph)
- "I need a graph where all the instances of :Taxpayer have a string value for their name and an integer value for their age and all their dependents are instances of :Dependent"
- Constraint are independent of each other
- Constraints don't need to cover entire graph
- Most constraints on RDF graphs target members of a class
- Generally simple to validate constraints on RDF graphs (model checking)
  - *But very difficult if data is an OWL KB*

Constraint View of Validation

Data

```
:Mary rdf:type :Taxpayer;
  :name "Mary Smith";   :age 37;
  :dependent :Chris, :Jeff;
  :spouse :Chris;
  :SSN 123456789 .
:Chris rdf:type :Person;
  :name "Chris Smith";  :age 36;
  :SSN 234567890 .
:Jeff rdf:type :Dependent;
  :name "Jeff Smith";   :age 9;
  :gender male .
:Susan rdf:type :Person;
  :name "Susan Smith" .
:Mary rdf:type :Person .    (!)
:Jeff rdf:type :Person .    (!)
```

Need to have typing available, either explicitly, or from inference against a simple ontology, or via some other means

Constraints

```
:Person ⊑ =1:name & ∀:name xsd:string
        & =1:age & ∀:age xsd:int
:Dependent ⊑ :Person
        & =1:SSN & ∀:SSN xsd:integer
:Taxpayer ⊑ :Person
        & =1:SSN ∧ ∀:SSN xsd:integer
        & ≤6 :dependent
        & ∀:dependent :Dependent
```

NB: These are not OWL axioms as they work under a "closed world" assumption.

- Violations
  - *:Susan has no :age specified*
  - *:Jeff has no :SSN specified*
  - *:Chris does not belong to :Dependent*
  - ~~*:Mary might have more than 6 dependents*~~

Schema View of Validation

- Validation is like schema recognition (in, e.g., Relax NG), but unordered.
- "I need (will produce) a graph containing a node with type :Taxpayer and name a string and :SSN an integer and zero to ten dependents, each of which has a string name and integer :SSN and :age; and no other information"
- Call the things being recognized shapes
- Each shape component matches different links in the graph
  - *Just like each bit of Relax NG schemas match different parts of an XML document*
- Each link in graph must be matched by some shape component
- Can easily be difficult to determine matches even for RDF graphs
  - *Need to consider multiple ways to match*

Schema View of Validation

Data

```
:Mary rdf:type :Taxpayer;
  :name "Mary Smith";   :age 37;
  :dependent :Chris, :Jeff;
  :spouse :Chris;
  :SSN 123456789 .
:Chris rdf:type :Person;
  :name "Chris Smith";  :age 36;
  :SSN 234567890 .
:Jeff rdf:type :Dependent;
  :name "Jeff Smith";   :age 9;
  :gender male .
:Susan rdf:type :Person;
   :name "Susan Smith" .

:Mary rdf:type :Person .
:Jeff rdfs:type :Person .
```

Type links often not needed

Shapes

```
<Person> { rdf:type ( :Person ) ? ,
   :name xsd:string , :age xsd:integer }
<Taxpayer> { &<Person> ,
   rdf:type ( :Taxpayer ) , :SSN xsd:integer ,
   :dependent @<Person> {0,1},
   :dependent @<Dependent> {0,5},
   .-:dependent @<UniversalShape>* }
<Dependent> { &<Person> , :SSN xsd:integer }
<UniversalShape> { . @<UniversalShape>* }
```

- Matches
  - *Chris matches :Dependent but not :Person*
  - *Jeff does not match :Dependent or :Person*
    - Extra rdf:type and :gender, missing :SSN
  - *Mary does not match :Taxpayer*
    - Catchall matches :spouse, but not Jeff
    - NB: Can have up to 6 dependents!
  - *All links consumed*

Constraint View Semantics (OWL Variant)

- The meaning of an RDF graph is its Herbrand interpretation
  - *except that literals are treated as their values not their syntax*
  - *i.e., (roughly) the graph itself treated as facts*
  - *maybe after RDFS completion of the graph*
- This interpretation treats each blank node as different from other nodes
- Constraints are given their OWL Full (!) meaning
  - *A shape is satisfied if the OWL axiom is true in the Herbrand interpretation*
  - *Model checking with only one model — cheap!*

Schema View Semantics (One Variant)

- New semantics for RDF graphs (but quite standard)
  - *The meaning of an RDF graph is its graph structure*
  - *Each blank node is treated as different from all other nodes*
  - *Literals are their syntax (maybe)*
- A set of shapes is satisfied by an RDF graph if there is a mapping from nodes in the graph to shapes (or to non-empty sets of shapes) such that each node satisfies the/each shape that it is mapped to.
- Shapes are satisfied by a node if the node satisfies the local parts of the shape and its links can be assigned to non-local parts in a way that satisfies their numbers and the link values are mapped to their non-local shape
- Initial mapping from nodes to shapes looks costly
  - *Sometimes this can be done cheaply, but in many cases it is expensive*
- Can set initial requirements for mapping, e.g., Mary must map to TaxpayerShape

The Two Views Compared

Constraint View

- Shapes, containing
  - *constraints (both local and on property values)*
  - *target selectors*
- Constraints satisfied independently
- Targeting obviates need for recursion
- Shapes work on their targeted nodes
- Can ignore nodes and links
- Basically model checking the graph
  - *Inexpensive, easy to implement*

Schema View

- Named shapes, containing
  - *constraints (both local and on property values)*
- Constraints satisfied additively
- Recursive shapes needed
- Shapes match against nodes, consuming links
- Entire graph must be consumed
- Requires assigning nodes to shapes
  - *Can be expensive and hard to implement*

ISSUE: Recursive Shapes and Data Loops

Data

```
:i1 :p :i1 .
:i2a :p :i2b .
:i2b :p :i2a .
```

Shapes

```
<UniversalShape> {
    . @<UniversalShape>* }

<S1> { :p !@<S1> }
```

Mappings (NB: Two!)

```
M(:i1) = { UniversalShape }
M(:i2a) = { UniversalShape, S1 }
M(:i2b) = { UniversalShape }
```

```
M(:i1) = { UniversalShape }
M(:i2a) = { UniversalShape }
M(:i2b) = { UniversalShape, S1 }
```

Schema View

- Needs recursive shapes, e.g., UniversalShape
- Recursion works acceptably with simple constructs
- Recursion has problems with negation
  - *No useful mappings*
  - *Two different mappings — choose which one?*
  - *So no recursion through negation (and some other constructs)*

ISSUE: Recursive Shapes and Data Loops

Data

```
:i1 rdf:type :T1 .
:i1 :p :i1 .

:i2a rdf:type :T2 .
:i2a :p :i2b .
:i2b :p :i2a .

:i3 y :i2a .
```

Shapes

```
:T1 ⊑ ∀:p ! :T1
:T2 ⊑ ∀:p ! :T2
:i3 ∈ ∀y :T3
:T3 = ∀:p :T3
```

- Violations
  - *First shape: :i1*
  - *Second shape: none*
  - *Other shapes: none?*

Constraint View

- Doesn't need recursive shapes (nearly as much)
  - *No sub-shapes in SPIN Constraints or Stardog ICV*
- Adding sub-shapes is useful, but leads to recursion
- What is the meaning of simple recursion?
  - *No guidance from description logic constraints*
  - *Maximal satisfaction: looping is success*
- Recursion through negation is again a problem
- Potential solutions:
  - *No recursion at all ***
  - *No recursion through negation*
  - *Encountering a recursion loop is an error*

Analysis So Far

Two quite different views:

- Different basic motivations: constraints vs schemas
- Different triggering: targetting vs mapping/matching
- Different results: violations vs matches
- Different formal semantics
- Differential need for recursive shapes
- Different expressive power
  - *Not all that different if certain extensions are made on either side*
- Different complexity: easy vs mostly difficult

Schema-Constraints Reconciliation?

- Reconciliation not really possible
  - *Big difference between conjunctive and additive*
  - *Schema view has complex and difficult validation*
    - Additive constructs need to make choices
    - There are easy cases but these are essentially the conjunctive ones
- Schema view needs recursive shapes because there is no targetting
  - *Could add targetting but that would be even more complex*
- How about adding some limited additive pieces to constraint view?
  - *Already done in counting constructs*
  - *Could add a partitioning component, but that can easily result in difficult validation*
  - *Adding global coverage would also result in difficult validation*

THE FIRST DECISION

Choose between constraint view and schema view

Outcome: SHACL is based on the constraint view of validation

- Constraint view has commercial implementations: SPIN Constraints, Stardog ICV
- Constraint view has easier implementation
- Constraint view is better understood?
  - *More on this later*

SHACL

SHACL Shape

```
s:s1 rdf:type sh:Shape ;
  sh:targetClass :Person ;
  sh:nodeKind sh:IRI ;
  sh:stem "http://p.google.com/";
  sh:property [
    sh:predicate :age ;
    sh:minCardinality 1;
    sh:maxCardinality 1;
    sh:datatype xsd:integer ] ;
  sh:property [
    sh:predicate :name ;
    sh:uniqueLang true ;
    sh:datatype xsd:string ;
    sh:minLength 5 ] ;
  sh:property [
    sh:predicate :child ;
    sh:class :Person ;
    sh:shape [ a sh:Shape ; ... ] ];
  sh:property [
    sh:path ( :child :age );
    sh:lessThan :age ] .
```

Shapes with

- targeting of:
  - *all members of a class (OWLish?)*
  - *a particular node (OWLish)*
  - *all subjects of a property (OWLish)*
  - *all objects of a property (OWLish)*
- constraints (local or all path values):
  - *class membership (OWLish?)*
  - *datatype membership (OWLish?)*
  - *IRI vs blank node vs literal*
  - *membership in a list (OWLish)*
  - *comparing vs a constant (OWLish)*
  - *string length and regex match (OWLish for literals, not IRIs)*
  - *validate against another shape (OWLish if non-recursive)*
- constraints for path(s):
  - *number of values for path (OWLish)*
  - *path has a particular value (OWLish)*
  - *unique value for each language (could be OWLish)*
  - *same/different set of values for two paths (could be OWLish)*
  - *compare value sets for two paths (could be OWLish)*
- boolean combinations (OWLish)
- no values for unmentioned properties
- recursion (but not currently)

Relationship Between RDF(S) and SHACL

SHACL uses RDF literals and RDF datatypes

- But what are literals and datatypes in SHACL?
  - *Values and sets, as in RDF Semantics (OWLish)*
  - *Pieces of syntax, as in RDF graph*
  - *Either, depending on the construct (SPARQLish) ***

SHACL uses class membership (as does RDF and RDFS)

- But what is class membership in SHACL?
  - *What RDFS says: in class extension in RDFS interpretations (OWLish)*
  - *What RDF says: direct rdf:type link to class (can be OWLish)*
  - *Something else: e.g., rdf:type/rdfs:subClassOf* path to class (SPARQLish) ***

SPARQL Semantics for SHACL

- Some SHACL constructs don't fit well into OWL semantics
- So use SPARQL as basis for SHACL?
  - *Meaning of SHACL is specified by translation to SPARQL*
  - *Each shape becomes a SPARQL query*
  - *Results of queries become violations*
- Can extend SHACL by using SPARQL code directly

```
s:s2 rdf:type sh:Shape ;
 sh:targetClass :Person ;
 sh:nodeKind sh:IRI ;
sh:property [
 sh:predicate :child ;
 sh:class :Person ;
 sh:shape s:s3 ] .
s:s3 rdf:type sh:Shape ;
 sh:nodeKind sh:blankNode.
```

```
SELECT ?this WHERE {
 ?this rdf:type/rdfs:subClassOf* :Person .
 FILTER ! ( isIRI(?this)
  && ( ! EXISTS { ?this :child ?that .
        FILTER NOT EXISTS {
       ?that rdf:type/rdfs:subClassOf :Person.} } )
  && ( ! EXISTS { ?this :child ?that .
      FILTER NOT EXISTS {
        SELECT ?that WHERE {
          FILTER ! ( isBlank(?that) ) } } } ) )
}
```

OWL-SPARQL Semantics Reconciliation

- OWL semantics can be implemented in SPARQL [4,5]
- Is it possible to have OWL semantics as normative and SPARQL semantics as an implementation method?
  - *Not easily*
    - SPARQL-only bits (SPARQL class membership, looking at syntax of literals, looking at IRIs, unmentioned properties)
    - recursion (no recursion now, but desire still remains)
  - *Extending OWL semantics to most of these is possible but difficult [5]*
- SPARQL semantics can get effect of RDFS semantics by adding the RDFS inferences to an RDF graph

THE SECOND DECISION

Choose between OWL semantics and SPARQL semantics

Outcome: SHACL's semantics are via translaion to SPARQL

- SPARQL semantics is more flexible
- SPARQL semantics provides direct implementation path
- SPARQL semantics easily handles things like
  - *checking IRI vs blank node vs literal*
  - *requiring IRIs to have a certain prefix*
  - *requiring particular datatypes for literals*
  - *unmentioned properties have no values*
- SPARQL semantics can handle recursion ?

Another SPARQL Translation for SHACL

- SPARQL translation sketched above can't handle recursion
- A different approach is to instead translate a shape to multiple queries and use some "glue" code
  - *Separate query to select the target nodes*
  - *Separate query for each constraint of each shape*
  - *External glue says how target results initiate constraint queries*
  - *Connection to sub-shapes employs SPARQL function call glue*
  - *Glue has to be inside SPARQL so that blank nodes are preserved*

Another SPARQL Translation (Simplified)

```
s:s2 rdf:type sh:Shape ;
 sh:targetClass :Person ;
 sh:nodeKind sh:IRI ;

 sh:property [
  sh:predicate :child ;
  sh:class :Person ;

  sh:shape s:s3 ] .


s:s3 rdf:type sh:Shape ;
 sh:nodeKind sh:blankNode.
```

```
t:  SELECT ?this WHERE {
     ?this rdf:type/rdfs:subClassOf :Person . }
c1: SELECT $this WHERE {
     FILTER ( ! isIRI($this) ) }
c2: SELECT $this WHERE {
     $this :child ?value .
     FILTER NOT EXISTS {
      ?value rdf:type/rdfs:subClassOf :Person . }}
c3: SELECT $this WHERE {
     $this :child ?value .
     FILTER NOT EXISTS sh:hasShape(?value,c4)
c4: SELECT $this WHERE {
     FILTER ( ! isBlank($this) ) }
```

- Controlling code needs to feed results of target query to constraint queries
- Constraint queries are run with $this pre-bound
  - *pre-binding is widely implemented, but poorly defined*
- sh:hasShape is a glue function that runs the named query
  - *Deep inside SPARQL processing but calls other SPARQL queries*
  - *Results in a top-down order of evaluation, i.e., different from SPARQL*

THE THIRD DECISION

Choose whether SPARQL translation should use extensions to SPARQL

Outcome: SHACL uses extensions to SPARQL

- pre-binding, sh:hasShape, and more
- Using extensions permits recursion
- Using extensions produces small SPARQL queries
- Issues:
  - *Requires SPARQL function that calls SPARQL queries*
  - *Results in visible top-down evaluation order*
  - *Requires use of pre-binding*
  - *Each query is run multiple times*

Extending SHACL with SPARQL

- So far SPARQL is just used to provide the meaning for SHACL constructs
- SHACL is not a universal language, so allow (parameterized) SPARQL code
  - *Anything that can be done in SPARQL is part of SHACL*
- Extension exposes translation to SPARQL

Direct Usage

```
s:sc rdf:type sh:Shape ;
 sh:sparql [ sh:select
  """SELECT $this ?value WHERE {
     $this $PATH ?value
     FILTER NOT EXISTS {
       $this rdf:type :Person } }"""
 ].
```

Parameterized Usage

```
s:sc rdf:type sh:Shape ;
  se:directClass :Person .
```

Parameterized Definition

```
se:directClassComponent
 rdf:type sh:ConstraintComponent ;
 sh:parameter [
  sh:predicate se:directClass ;
  sh:nodeKind sh:IRI ;
  sh:description "Direct class" ] ;
 sh:propertyValidator [ sh:select
  """SELECT $this ?value WHERE {
     $this $PATH ?value .
     FILTER NOT EXISTS {
      $this rdf:type $class } }""" ] ;
 sh:shapeValidator [ sh:select
  """SELECT $this ?value WHERE { ... }""" ].
```

Recursion Revisited

- SHACL now has little need for recursion
  - *Target selection and classes replace recursion in many cases*
- Recursion opens up a form of recognition, not checking

```
:polentone rdf:type sh:Shape ;
  sh:property [
    sh:predicate :birthPlace ;
    sh:class :ItalyNorthOfPo ]
  sh:property [
    sh:predicate :knows ;
    sh:shape :polentone ] ;
```

- Translation to SPARQL has been chosen in part because it can support recursion
- Removing recursion would allow different SPARQL translations
  - *Implement SHACL on top of unmodified SPARQL implementations*
  - *No need for pre-binding*
  - *Validate RDF graphs accessible only via SPARQL endpoints*

Problems Using SPARQL for/in SHACL

- Core of SHACL, a simple language, depends on a large and complex language
- SPARQL definition [8] has problems that affect SHACL
  - *No definition for pre-binding so SHACL needs its own*
    - Every definition for pre-binding in SHACL hasn't worked
  - *EXISTS in SPARQL has errors and counter-intuitive aspects*
    - Counter-intutive combination with blank nodes, MINUS, subqueries
    - Implementations do not follow definition
    - Implementations differ
  - *See poster at ISWC [9]*
- To be viable, SHACL needs fixes to SPARQL
  - *These fixes are slow in coming*

Current Status of SHACL

- Complete core language (maybe an addition or two to be done)
  - *No recursion*
- Complete extension mechanism (very complex)
  - *Not only shapes and constraints but also functions and more*
- Meaning of SHACL given as translation to SPARQL (broken)
  - *Uses a form of pre-binding (broken)*
  - *Uses an extension function (questionable)*
  - *Imposes a top-down execution order (inefficient)*
- SPARQL used as SHACL extension mechanism (broken)
  - *Exposes above problems to SHACL coders*
- Lots of other problems remain in SHACL document

Difficulties in Producing SHACL

- Clash between several groups with divergent views of SHACL
    1. *Schemas vs constraints*
    2. *SPARQL vs non-SPARQL semantics*
    3. *Just SPARQL or SPARQL plus extensions*
    4. *Need for formality (not addressed earlier)*
    - *Groups did not converge, and actually appeared to diverge*
- Too few active members in Working group
    - *Many differences were between just two people*
- Insufficient expertise or interest from initial editors
- Problems with SPARQL (EXISTS, pre-binding)
- Little buy-in from commercial SPARQL implementors
    - *Essentially only TopQuadrant*
    - *Implementation only at candidate recommendation stage is not helpful*

References

1. **Shapes Constraint Language (SHACL):** https://www.w3.org/TR/shacl/
2. **Shape Expressions:** https://www.w3.org/2001/sw/wiki/ShEx
3. **Shape Expressions Semantics:** Slawek Staworko, Iovka Boneva, Jose E. Labra Gayo, Samuel Hym, Eric G. Prud'hommeaux, and Harold Solbrig. "Complexity and Expressiveness of ShEx for RDF". *ICDT 2015*.
4. **Description Logic Constraints:** J. Tao, E. Sirin, J. Bao, and D. L. McGuinness. "Integrity constraints in OWL". *AAAI-2010*.
5. **Description Logic Constraints for RDF:** Peter F. Patel-Schneider. "Using Description Logics for RDF Constraint Checking and Closed-World Recognition". *AAAI-2015*.
6. **SPIN Constraints:** http://www.topquadrant.com/technology/sparql-rules-spin/spin-constraints/
7. **Stardog ICV:** http://docs.stardog.com/icv/icv-specification.html
8. **SPARQL 1.1 Query Language:** https://www.w3.org/TR/sparql11-query/
9. **SPARQL EXISTS problems:** Peter F. Patel-Schneider and David Martin. "EXISTSential Aspects of SPARQL". *ISWC 2016*.