Constraint Validation with OWL 2 — Closed World Semantics — With and Without Reasoning

XXXXX[1] and XXXXX[2]

[1] XXXXX
XXXXX,
[2] XXXXX
XXXXX

Abstract. OWL (and RDFS) semantics (and syntax) can be used to specify constraints.

Keywords: ..

1 Introduction

2 OWL as Constraint Validation Language

2.1 Compact, Human-Readable Syntax

1 (2) example(s)

1. SPARQL
2. SPIN (Turtle syntax)
3. OWL 2 (Turtle syntax)
4. OWL 2 (Manchester Syntax)

cardinality restrictions

for the simple cases the sytax in compact form and manchester are not that different. As well as showing that the RDF versions are quite close as well.

In Shex Compact (corrections welcome),

```
<WebServicePersonShape> {
  rdf:type foad:Person ,
  foaf:name xsd:string ,
  foo:email xsd:string ,
  foo:phone xsd:string *
}
```

In ShEx RDF

```
<WebServicePersonShape> a rs:ResourceShape ;
     rs:property  [ rs:name                "one name" ;
                    rs:occurs              rs:Exactly-one .  ;
                    rs:propertyDefinition  foaf:name ;
                    rs:valueShape          foaf:Person
                   ],
                   [ rs:name                "one e mail" ;
                    rs:occurs              rs:Exactly-one ;
                    rs:propertyDefinition  foo:email ;
                    rs:valueShape          foaf:Person
                   ] ,
                   [ rs:name                "some number of phone numbers" ;
                    rs:occurs              rs:Zero-or-many ;
                    rs:propertyDefinition  foo:phone ;
                    rs:valueShape          foaf:Person
                   ] ;
```

In spin turtle

```
foaf:person a owl:Class;
  spin:constraint [ rdfs:label "one name" ;
                    a spl:Attribute ;
                    spl:predicate foaf:name ;
                    spl:count 1 ] ,
                  [ rdfs:label "one e-mail" ;
                    a spl:Attribute ;
                    spl:predicate foo:email ;
                    spl:count 1 ] ,
                  [ rdfs:label "some number of phone numbers"
                    a spl:Attribute ;
                    spl:predicate foo:phone ;
                    spl:minCount 0 ] .
```

–¿ As given it looks horrible, but with a compact syntax on top
In Closed world owl rdf turtle serialisation

```
foaf:person a owl:Class;
  rdfs:subClassOf [ a       owl:Restriction ;
                    owl:cardinality "1"^^xsd:nonNegativeInteger ;
                    owl:onProperty foaf:name ;
                    rdfs:label "one name"
                  ] ,
                  [ a       owl:Restriction ;
                    owl:cardinality "1"^^xsd:nonNegativeInteger ;
                    owl:onProperty foo:email ;
                    rdfs:label "one email"
                  ] ,
                  [ a       owl:Restriction ;
                    owl:minCardinality "0"^^xsd:nonNegativeInteger ;
                    owl:onProperty foo:phone ;
                    rdfs:label "some number of phone numbers"
                  ] ,
```

Some Manchester syntax (again corrections welcome)

```
Class: foaf:person
   foaf:name exactly 1
   foo:email exactly 1
   foo:phone min 0
```

which existing technology should be used as guide. As well as showing what the technologies in the field can do already.

syntactically well-formed email addresses

Thomas: example

Yes, your specific example can be nicely represented in a very compact textual form. But your example only talks about cardinality, and there is obviously much more that people want to specify when they publish data models. For example you may want to say that the values of foo:email are syntactically well-formed email addresses. Simple languages quickly reach limits and then need to be overloaded with all kinds of escape mechanisms.

RDF is designed to be extensible - anyone can add their own classes and properties to represent any kind of extra information. Therefore a model that talks about other data models should also be represented in RDF, and that's where Shapes come into the picture. For certain common shapes (such as cardinality) a short-hand syntax could be added, but still the underlying semantics need to be more flexible.

JSON-LD.

example:

```
{
    "@context" : "http://w3.org/2014/shapes/context",
    "@id" : "ex:Person",
    "constraint" : {
        "type" : "shapes:Property",
        "property" : "ex:firstName",
        "occurs" : "shapes:Exactly-one",
        "valueType" : "xsd:string"
    }
}

```

benefits:

- understandable by anyone who does some coding, without being a ”semantic web engineer.”

Why compact human-readable syntax? don't think RDF, don't know RDF, they don't want to know RDF. They don't think open world or closed world, etc.

Writing validation rules in RDF would be simpler than writing them in SPARQL, but still much more tedious than having a concise, compact syntax for expressing them.

requirements on compact human-readable syntax.

- understandable by anyone who does some coding, without being a ”semantic web engineer.”

- Now the question is: who creates this and how? And do they have to be fully versed in RDF?

structure of syntax. The structure itself should guide novices to make good RDF-compatible decisions.

expressing constraints in OWL enables thinking in RDF. — syntax using the same paradigm and semantics of RDF [important!]

We need a syntax that is related to RDF, uses the same paradigm and semantics.

I've seen RDF/XML and, to a lesser extent, JSON-LD act as gateways to RDF. (I almost wanted to say: gateway drugs.) The down side of that is that users tend not to learn to think in terms of key RDF concepts like graphs, inferencing, and the meaning of class membership in RDF. So it does help that there are known serializations, but those known serializations can hide the ways that RDF semantics differ from our previous ways of thinking about data. Therefore people tend to create perfectly good XML or JSON data that isn't RDF at heart.

My concern is that we had this confusion before: When RDF and OWL were standardized, they came with RDF/XML syntax only, and people thought this semantic web is just another kind of XML. But a complicating kind of XML, where the idea of graphs was hidden behind rdf:ID and rdf:about attributes.

From the data model perspective, RDF was never a kind of XML. It can be serialized as XML, but it has a different data model. This difference was, perhaps, obvious to the standard committee and they never thought anyone would be confused or that this confusion would be an issue.

In reality, this was, however, a big issue. It was very difficult for other people to understand what RDF was without a lot of education that had to change the assumptions they quickly made after first getting to know about RDF. First impressions are hard to change. Things became easier after Turtle became better known as the way to serialize and RDF/XML faded into background as just one (lesser used) serialization.

2.2 Readability in the face of complexity and internationalization of human friendly syntaxes

use of templates for better readability and management of complexity

complexity needs to be managed not ignored.

example in a number of syntaxes.

In ShEX Compact

```
<WebServicePersonShape> {
  rdf:type foaf:Person ,
  foaf:name xsd:string ,
  foo:email xsd:string ,
  foo:phone xsd:string *
}
```

In ShEx RDF

```
<WebServicePersonShape> a rs:ResourceShape ;
   rs:property  [ rs:occurs              rs:Exactly-one .  ;
                  rs:propertyDefinition  foaf:name ;
                  rs:valueShape          foaf:Person ],
                 [ rs:occurs              rs:Exactly-one ;
                  rs:propertyDefinition  foo:email ;
                  rs:valueShape          foaf:Person ] ,
                 [ rs:occurs              rs:Zero-or-many ;
                  rs:propertyDefinition  foo:phone ;
                  rs:valueShape          foaf:Person ] .
```

In SPIN turtle

```
foaf:person a owl:Class;
spin:constraint [ spl:predicate foaf:name ;
                  spl:count 1 ] ,
                [ spl:predicate foo:email ;
                  spl:count 1 ] ,
                [ spl:predicate foo:phone ;
                  spl:minCount 0 ] .
```

Now we change the requirements very slightly. Instead of any string being acceptable as e-mail we need to ensure it is actually a valid e-mail for an employee in our organisation.

How would you determine what is a valid e-mail for an employee in a organisation? The first approach is to say that it has to match a specific regex. e.g. for the university I went to any e-mail address of an employee needs to match the regex

`\.+@pl.hanze.nl^"`

.

Let's try that in ShEx compact shall we.

```
<WebServicePersonShape> {
  rdf:type foaf:Person ,
  foaf:name xsd:string ,
  foo:email IRI %sparql{ ?s foo:email ?mbox . FILTER (REGEX(str(?mbox), \.+@pl.hanze.nl^") %}
  foo:phone xsd:string *
}
```

And in spin turtle

```
foaf:person a owl:Class;
spin:constraint [ spl:predicate foaf:name ;
                  spl:count 1 ] ,
                [ spl:predicate foo:email ;
                  spl:count 1 ] ,
                [ sp:text \ASK{?s foo:email ?mbox . FILTER (!(REGEX(str(?mbox), \.+@pl.hanze.nl^")) }"
                [ spl:predicate foo:phone ;
                  spl:minCount 0 ] .
```

This can be simplified again with a template library e.g. similar to of the ones given at [1].

```
our-company:employee-email
      a       spin:Template ;
      rdfs:comment "This template check a IRI to be a valid employee e-mail property ?arg1."@en ;
      rdfs:label "syntax check in all instances:  valid employee email"@en ;
      spin:body
              [ sp:text """CONSTRUCT {
                   _:b0 a spin:ConstraintViolation .
                   _:b0 spin:violationRoot ?s .
                   _:b0 spin:violationPath ?arg1 .
                }
                WHERE {
                 ?arg1 foo:email ?mbox .
                 FILTER (!(REGEX(str(?mbox), \.+@pl.hanze.nl^"))
                }""" ;
```

So now the spin turtle becomes

```
foaf:person a owl:Class;
spin:constraint [ spl:predicate foaf:name ;
                  spl:count 1 ] ,
                [ spl:predicate foo:email ;
                  spl:count 1 ;
                  a our-company:employee-email] ,
                [ spl:predicate foo:phone ;
                  spl:minCount 0 ] .
```

Which is more likely to be readable to the “business” users.

The second advantage of this kind of approach is that these templates are maintainable independently from the rules themselves.

This becomes even more the case when the validation starts to be more complicated. Few companies have a clean separation between e-mails addresses for employees and mailing lists. However, most do have a LDAP or equivalent system. Assuming Squirrel RDF for LDAP[2] is still working a system can adapt that as an information source.

Shex now turns into this.

```
<WebServicePersonShape> {
  rdf:type foaf:Person ,
  foaf:name xsd:string ,
  foo:email IRI %sparql{ ?s foo:email ?mbox . SERVICE<ourcompanyLdap>{ ?employee foaf:mbox ?mbox ; a :Employee }) %}
  foo:phone xsd:string *
}
```

While the SPIN solution stays like this.

```
foaf:person a owl:Class;
spin:constraint [ spl:predicate foaf:name ;
                  spl:count 1 ] ,
                [ spl:predicate foo:email ;
                  spl:count 1 ;
                  a our-company:employee-email] ,
                [ spl:predicate foo:phone ;
                  spl:minCount 0 ] .
```

The SPIN solution can be made more compact if it had more predicates e.g.

```
foaf:person a owl:Class;
shape:hasOne  [ spl:predicate foaf:name  ] ,
              [ a our-company:employee-email ] .
shape:some    [ spl:predicate foo:phone ] .
```

Compare this to the compact ShEx form? Is the SPIN one that bad?

2.3 define vocabulary terms and constraints with the same syntax

If you want to properly publish a ”schema” then you also need a way to define the properties and classes that are used, together with their rdfs:labels, rdfs:comments, relationships etc. These are attached to the properties globally, and already exist in triple format. Developers would end up using a Turtle/JSON-LD file for one part of their schema, and then a custom syntax just for the constraints?

3 Class-Based Validation

Constraints can be associated with instances of RDFS/OWL types. I view the ability to associate constraints with instances of types as the most important aspect of a constraint system.

Thomas: how is it done using our approach

3.1 modularization of constraints / class-based validation vs. shapes-based validation

constraints can be hosted by a ”Shape” object or to an OWL class.

class-based validation

If we want to be based on RDF, then we should honor its basic principles. One of those principles is that of self-describing data, where applications can look up the definition of a resource by resolving URIs. For example, if a resource declares an rdf:type, then an application should be able to look up the definition of the class, including its semantic declarations and constraints. This should be the default assumption: someone creates an ontology and when he publishes it on the web then anyone who wants to reuse that ontology should honor the same constraints.

Having said this, applications may overload the default behavior. In TopBraid we do this by first looking at local files for the same URI, and preferring those over the files from the web. This also makes it possible to refer to graphs that do not even exist on the web.

shapes-based validation

When resources do not even have to have an rdf:type - how is this going to be self-describing data? What is the entry point into the validation?”

- It is self-describing because the Shape is either linked to the service description or the instance document.

- The entry point is the entire RDF document (aka graph) just like in SPARQL. There is not necessarily a special node in the graph. You use a general graph pattern to either find a subgraph that must exist (an assertion), or to find a subgraph that must not exist (a constraint violation).

This is motivated by Linked Data API considerations. Here one deals with REST services. For example, when creating new resources, one POSTs an RDF document to some URL. We want to describe the API contract for the service. The service is described by a Service Description resource. The Service Description resource refers to the Shape accepted by the POST request. Shape may require that the request contains a node with an rdf:type property. However, the Shape may apply to several different rdf:types. The Shape is therefore attached to the service, not each type. After the resource is POSTed, the service may added new triples, so the Shape of the created resource is different than the Shape of the POST request. The created resource may contain a link to its Shape. A service that lets you query across a set of resources may also link to a set of Shapes that define the data available for querying. This metadata both tells you what properties are available and may let you define more efficient queries.

The above use cases are orthogonal to use cases where an RDF class defines constraints that must be universally obeyed, independent of the context, i.e. a bicycle must have two wheels.

We have seen many more examples of the latter than the former, and I believe the structure of existing ontology languages suggests that it should be possible to "attach" constraints to classes. I personally believe your use case is the exception, not the rule, but of course I would like to see a generic solution that covers all use cases so that the standardization process succeeds.

Well if you only consider constraints for objects of properties, then it may turn out that constraints associated with an RDF class are in the minority. Of course, I strongly feel that constraints for objects of properties is not a central aspect of RDF validation.

3.2 Pros

In our experience, having an rdf:type is the norm, and typeless resources are the exception. I don't think it is a good idea to base the design of such an important aspect on the exception cases. Attaching the semantics to classes gives you a natural way of encapsulation, inheritance and allows us to reuse all existing ontologies in an incremental way, rather than coming up with a parallel world of free-floating shapes. - Object-orientation is mainstream. - (simple and consistent) "object-oriented" design used by all other ontology languages. In our experience, using rdf:type triples and classes is a more intuitive mechanism, and it already exists, is consistent with OWL etc.

Yes: RDF is all about graphs, but does this mean that a constraint language may not exploit certain triples from those graphs and give them special meaning?

Another analogy is XML, where every XML element also has a name. This is comparable to its rdf:type.

If we are talking about Linked Data and want to stay aligned with the way that ontologies are currently being built, then relating constraints to classes remains a requirement.

3.3 Cons

An RDF document is a set of triples (statements). The triples do not necessarily have to look like they came from an object (in the OO sense), i.e. that there is one distinguished root node (this, self) that has some type (class) attribute, and other triples (properties) that have the root node as their subject, and so on recursively.

If you think of an RDF document as a graph, then there should be no special significance to a node that happens to have a type attribute. We should be able to state constraints on the graph in the absence of type triples. This is very consistent with SPARQL. SPARQL includes a graph pattern matching language. SPARQL queries are not associated with any specific type, so why should constraints be?

However, even if the RDF document was the representation of some OO-like object that had a root node and some type (class) attribute, its shape could vary depending on the context. For example, when you create a resource using POST, certain triples will be absent, e.g. creation date, and if present they may be ignored or the server may fail the request. When you GET that resource you will normally see other triples that the server added, e.g. creation date, and these will occur exactly once.

OSLC lets you associate shapes with operations. Resource instances may link to shape resources that describe the instance. There is no need to base the association between resources and constraints using an RDF type. A shape could apply to several types or to no types.

4 Inferencing as a pre/post validation step — Reasoning and Constraint Checking

Both should be possible:

- Constraint Checking with Reasoning
- Constraint Checking without Reasoning

4.1 our approach

Existential Quantification on Object Properties

constraint (formal syntax):

```

```

constraint (Manchester syntax):

```

```

constraint (functional-style syntax):

```
SubClassOf( ObjectSomeValuesFrom( :hasDog owl:Thing ) :DogOwner )
```

constraint (turtle syntax):

```
    _:x
        a owl:Restriction ;
        owl:onProperty :hasDog ;
        owl:someValuesFrom owl:Thing ;
        rdfs:subClassOf :DogOwner .
```

data (Turtle syntax):

```
    :Peter
        a owl:Thing , owl2spin:ToInfer ;
        :hasDog :Brian .
    :Brian
        a owl:Thing .
```

inferred triples (Turtle syntax):

```
:Peter a :DogOwner .
```

4.2 Constraint Checking with Reasoning

Constraint validation does not appear to be part of the services provided by OWL. This has lead to claims that OWL cannot be used for constraint validation. However inference, which is the core service provided by OWL, and constraint validation are indeed very closely related.

Inference is the process of determining what follows from what has been stated. Inference ranges from simple (students are people, John is a student, therefore John is a person) to the very complex. Inference can also recognize impossibilities (students are people, John is a student, John is not a person, therefore there is a contradiction). In the presence of complete information, nothing new can be inferred, so inference only checks for impossibilities, i.e., constraint violations. So the way do constraints in OWL is to first set up complete information, and then just perform inference [this is related work].

For example, with the concept axiom

```
  ex:Person <= =1 foaf:name & all foaf:name xsd:string &
               =1 foaf:phone & all foaf:phone xsd:string &
               =2 ex:child
```

and in the presence of (locally) complete information, such as

```
  <John> in ex:Person .
  <John> foaf:name "John"^^xsd:string .
  <John> foaf:phone "+19085551212"^^xsd:string .
  <John> ex:child <Bill> .
  <John> ex:child <William> .
  <John> in <=1 foaf:name .
  <John> in <=1 foaf:phone .
  <John> in all child {<Bill>, <William>, <Susan>} .
  <Bill> /= <William> .
  <Bill> /= <Susan> .
  <Susan> /= <John> .
```

determining whether ¡John¿ belongs to ex:Person is just constraint validation.

Setting up complete information is just what was done above. In a model there is complete information, so considering an RDF graph (plus consequences) as a model turns OWL inference into constraint validation. Of course, this doesn't mean that you have to implement OWL inference in a model the same way that you need to with incomplete information. In fact, as above, constraint validation can be implemented as SPARQL queries.

Conclusion

In fact the main difference between recognition and constraint checking is that the former either has no axioms or only uses axioms defining names that do not occur in the RDF graph whereas constraint checking uses axioms that relate concepts appearing in the RDF graph to descriptions.

So OWL can indeed be used for both the syntax and semantics of constraint checking and closed-world recognition in RDF, and most or all of it can be implemented using a translation to SPARQL queries.

—–

1. inference –¿ complete model
2. constraint checking

—–

Mentioning the use of inferencing as a pre/post validation step option is perfectly fine, enforcing it is a completely different thing.

Well it's not as simple as just mentioning that inferencing is a pre/post validation step option, depending on how your validation process is defined. To make validation play well with inference you may have to be able to handle infinite sets and multiple models on one side and too-central-to-turn-off inferences on the other.

use structural information for inferencing

- use structural information to produce new output (e.g. XML trees or other RDF triples) (see ShEx paper)

rules

- Another example is spin:rule, which is in our experience tremendously useful for defining mappings between ontologies, and to calculate the ex:area of a ex:Rectangle from ex:width and ex:height. Once we have a mechanism to attach SPARQL and templates to classes for constraint checking, we could use exactly the same mechanism to define such production rules.

sub-class super-class inferencing

Constraint definitions attached to a superclass (assuming Shape=class) need to also apply to instances of a subclass.

4.3 Constraint Validation Without Reasoning

5 Context-Sensitive Constraint Validation

main assertions:

– Reasoning and Closed World Assumption
– constraints depend on the context

But without reasoning things can work very well as well.

reasoning would either hide the error

There is an underlying practical problem, which we face daily in the management of the LOV repository. Again, it's not a minor point that RDF uses URIs for classes and properties. If you find a class or property URI somewhere in data (e.g., DBpedia) how do you figure the semantics of this URI, IOW how do you find the document(s) you are speaking about which define it, and which ones will you abide by, to figure in practice the closure of the description of this URI? Look at the result of this query http://bit.ly/UPGA25 ran on current LOV database of 450 vocabularies/ontologies. It yields all elements related to foaf:Person in various namespaces (and so many documents). Somehow each of those 300+ assertions is refining the semantics of foaf:Person. If you ignore the 25 ones defined by the foaf: namespace itself, you are lefy with more than twenty different namespaces/vocabularies/documents to explore, and select or not to figure the closure.

Of course, looking only at the FOAF vocabulary using a HTTP GET on foaf:Person, you will not access and even be aware of all this extra semantics, and you can ignore it, apart of those which FOAF itself reuses.

This is an extreme case maybe, but it illustrates what the practical problem will be, when using a URI, to first find out to which semantics you abide by before passing it to reasoners or constraint checkers. One could imagine something like a content negotiation on the URI where you pass in parameters the type of description you want for this URI (give me SPIN, OWL, Shapes ...). But given the messy way content negotiation is already implemented after httpRange14 resolution (based also on our LOV experience), I'm very skeptical about such a solution in practice. Moreover, different parties will use different documents of which the original URI owner does not control, is not aware of, let alone agrees with.

My concern is to see a door wide open to more balkanization of the Semantic Web, each tribe abiding by its own, not necessarily public, interpretation of shared vocabularies.

So if you use RDFS and OWL, and include a lot of constraints, then the terms are not very reusable because they carry a lot of baggage in the form of inferences which may not make sense in your application.

Note that I am describing the state of the practice. Many ontologies in fact do contain a lot of inferences and this makes them less reusable. This is true even in absence of constraints since W3C OWL doesn't officially handle constraints, just inference rules.

The point I was trying to make is that applications often need to impose additional constraints on terms and that these constraints often depend on the context.

Let's refocus the discussion on how an application designer would reuse an externally published OWL ontology and impose additional constraints. Let's assume OWL IC semantics. What would the application designer provide to describe a REST API that required exactly one dcterms:title triple in the request body on POST?

5.1 entailment regimes

ShEx and Resource Shapes work on an RDF graph. This does not mean that they can be used in combination with a reasoner, even an RDF reasoner. SPARQL has entailment regimes, which specifies how SPARQL is supposed to work with reasoning.

Consider, for example, that there are infinitely many RDF consequences of a finite RDF graph. How do I run ShEx and Resource Shapes on an infinite graph?

Consider, for example, that the OWL consequences of a finite RDF graph are generally not representable as an RDF graph. How do I combine ShEx or Resource Shapes with OWL?

support of inferences from ontologies It may be that there are many ontologies that support a lot of inferences. However, there are lots of ontologies that are small and do not support a lot of inferences. RDFS and OWL can be used both ways.

Further, why should an ontology that supports many inferences be undesirable, even if it is less reusable? I don't think that you can argue that reusability is a good in and of itself. For example, the ontology containing only the axiom stating that owl:Thing is a subclass of owl:Thing is maximally reusable, but not very useful.

6 Comparison of Validation/Constraint Languages — Why OWL 2 and SPARQL for RDF Validation?

dependencies between validation languages

Here is a diagram draft illustrating the dependencies among various Semantic Web languages, as I see them:

https://twitter.com/HolgerKnublauch/statuses/494648159931346944

The stacking of boxes on top of each other means "uses features of". Please don't get caught up in details. The points that I am trying to make are:

- There can be multiple "schema" languages for the same RDF models

- Different languages have different design goals (e.g. OWL was mainly created for classification tasks)

- All these languages can live alongside with each other without breaking each others interpretations, for example via separate import mechanisms (owl:imports, spin:imports, ic:imports). A SPIN model can ignore OWL constructs, but SPARQL queries can also be executed on top of graphs that have OWL inferencing activated. Likewise, OWL models can ignore spin:constraint definitions.

To me one of the important things is to ensure that classes, properties and instances can be reused. The concept of "Classes" is quite basic, established and successful, and is therefore a good foundation for Shapes too.

And yes, I am fully aware that there are other languages that did not make it into this diagram. I just wanted to focus on the comparison between OWL and Shapes+SPIN and make clear that this is not an either-or discussion. There is plenty of space for both approaches, and indeed both "communities" can benefit from each other when the Semantic Web grows as a whole.

Why OWL SPARQL for RDF validation?

- OWL SPARQL are sufficient for validation in a CWA
- Normally, the RDF has already an OWL / RDFS. Thus, part of those declarations will be defined anyway
- In most cases reusing existing OWL schemas for validation is enough
- What we need in the end is tools that translate OWL to SPARQL - or to something intermediate like ShEx or SPIN
- language that many people already know
- in practice that most people out there make use of RDFS and OWL constructs to tell other people and applications about how valid instances should be created.Whether this is good or bad is another question, but that's the reality.
- OWL semantics: Most people don't care about open world semantics, but of course nobody would admit that because it's against the specs and thousands of academic papers.
- could be interpreted either with the open world assumption to support open world reasoning, and (exactly the same piece) interpreted in closed world applications as a constraint for interfaces or a validation rule

difference between a declarative validation / constraint language (e.g. OWL 2, ShEx) and a more general purpose validation / constraint language (e.g. SPARQL)

Why declarative validation / constraint language?

a human readable syntax is vastly preferable to an RDF based syntax

Why general purpose validation / constraint language?

- maximum expressivity - SPARQL is very expressive, so most typical constraints can be expressed - need to express more complicated constraints, and SPARQL is the best available language to represent those

Why NOT general purpose validation / constraint language?

too low-level: SPARQL is too low-level. If you look at a SPARQL query, you'd have a hard time figuring out what constraint it is checking.

Why OWL / OWL 2 for RDF validation?

Why NOT OWL / OWL 2 for RDF validation?

The problem with OWL is the different semantics so people have to rewrite it in another languages such as SPIN / ShEx / SPARQL

http://composing-the-semantic-web.blogspot.com.au/2010/04/where-owl-fails.html

intended for (1) semantic description (2) reasoning: infer new triples from a given set of triples not intended for (1) validation (2) describe a database structure

Why not use OWL to describe database structures? (example)

it is hard to read an OWL ontology to understand the structure of a database.

ShEx:

```
A {
  (ex:samepred1 @B | ex:samepred1 @E) #either referencing B or E
}
C {
  ex:samepred1 @D
}
```

OWL:

```
Class A -> ex:samepred1 -> Class B
Class C -> ex:samepred1 -> Class D
A <= all ex:samepred1 Extra
C <= all ex:samePred1 D
Class B -> rdfs:subClassOf -> Class Extra
Class E -> rdfs:subClassOf -> Class Extra
```

Here have an extra class 'Extra' for which it is not clear whether is created for making the ontology work or is an actual Concept in my database.

OWL has disjunction, so you can say A ¡= all ex:samepred1 (B or E) which is even better than the ShEx version you give.

(3) to be used as Schema language for validation (4) further reasons: conclusion of this paper (http://arxiv.org/pdf/1404.1270v1.pdf) (5) Another thing you can not do in OWL is to define the following:

```
Type A -> ex:samepred1 -> Type B
Type C -> ex:samepred1 -> Type D
```

If you would define this in OWL you will also get, because OWL is property oriented and a property can only define a range and domain

```
Type A -> ex:samepred1 -> Type D
Type C -> ex:samepred1 -> Type B
```

Which is something we do not want.

Why do you say that? OWL includes many constructs that are local. Here you can use allValuesFrom, as in

```
A <= all ex:samepred1 B
C <= all ex:samePred1 D
```

– Clark Parsia had previously proposed the ICV semantics for OWL, but this was not submitted to W3C.
– Ambiguousness when OWL would be used assuming both OWA (for inferencing) and CWA (for constraint validation) (see section XXXXX). It would be somewhat confusing to have two completely different semantics for OWL.
– OWL cannot represent all possible use cases

(6) modeling goals / orientation

SHEX is oriented on Shapes and Shape Properties (Property per shape), whereas OWL is oriented on (Restriction) Classes and Properties. Orientation on (Restriction) Classes and Properties is a logic and the best thing to do when working in the world of reasoning.

However if we look from user point of view, which would like to understand the structure of database or interface of a service then an orientation on Shape and Shape Properties is better. This is what a user needs when he wants to write some queries for a database as noted earlier on by me.

OWL vs. ShEx

OWL is very good at semantic level / Shape Expressions are more suited for the syntactic level or data integration level

classes are semantic / shapes are syntactic

OWL: inferencing

ShEx: looks at the shape of the RDF graph

Why SPIN?

combines the flexibility of SPARQL with the simple declarative syntax of RDF triples

SPIN is self-describing

is extensible to any constraint

grows with the evolution and adoption of SPARQL

7 Validation using Closed World vs. Open World Semantics — OWL Closed World + SPARQL for RDF Validation — Underlying Semantics / Assumptions: Closed World Assumption and Unique Name Assumption

Underlying Assumptions: Closed World Assumption and Unique Name Assumption

OWL descriptions and the OWL semantics provide the necessary framework for both validating constraints and providing recognition facilities. Why then are there claims that OWL is inadequate for these purposes? I do not know why, but there are several aspects of OWL that might not be consonant with constraints and the kind of recognition that might be desired. However, it turns out that both OWL syntax and semantics are arguably the right solution for RDF constraint checking and closed-world recognition.

Closed-World Recognition / class assertions

Determining whether an individual belongs to an OWL concept is recognition.

Let's first look at recognition. Recognition is the basic operation in ShEx - we want to determine whether a particular node in an RDF graph matches a ShEx shape, for example, to say that John in

```
<John> foaf:name "John"^^xsd:string .
<John> foaf:phone "+19085551212"^^xsd:string .
<John> ex:child <Bill> .
<John> ex:child <William> .
```

matches

```
{ foaf:name xsd:string , foaf:phone xsd:string, ex:child [2] }
```

Recognition is also a basic operation of OWL.

The OWL version of the above ShEx expression (using a version of the OWL-DL publication syntax) is

```
  =1 foaf:name & all foaf:name xsd:string &
  =1 foaf:phone & all foaf:phone xsd:string &
  =2 ex:child
```

So it seems that OWL can easily handle ShEx recognition.

However, John does not match the above OWL description. Why is this? It is precisely that OWL does not assume that absence of information is information about absence. RDF works under the same assumptions, by the way. John could have more than one name as far as the above information is concerned. OWL (and RDF too) also does not assume that different names refer to different individuals. Bill and William could be the same person.

OWL has facilities to explicitly state information about absence and information about differences. If we add

```
  <John> in <=1 foaf:name .
  <John> in <=1 foaf:phone .
   <John> in all child {<Bill>, <William>} .
  <Bill> /= <William> .
```

then John does match the above description.

So it is not that OWL does not perform the kind of recognition that underlies ShEx, it is just that OWL does not make the assumption that absence of information is information about absence.

However, suppose that we want to make this assumption. This is roughly equivalent to saying that a system assumes that if it can't determine some fact, then that fact is false. There is a very large body of work on this topic because there are many tricky questions that arise with respect to closure in any sophisticated formalism, and OWL is indeed sophisticated.

Fortunately RDF and RDFS are not very sophisticated at all, and the tricky questions just do not arise if information comes in the form of RDF and RDFS triples. The basic idea is to treat the triples (and their RDF and RDFS consequences if desired) as completely describing the world. So, 1/ if a triple is not present then it is false and 2/ different IRIs describe different individuals. This is precisely the same idea that underlies model checking. First-order inference is undecidable, but determining whether a first-order sentence is true in one particular state of the world is much, much easier.

So it is possible to use the OWL syntactic and semantic machinery to define how to recognize OWL descriptions under just the same assumptions that underlie ShEx. The only change from the standard OWL setup is to define how to go from an RDF graph to an OWL model. (There are some technical details that interfere with this general description of the account, but they are easy to handle.) Definitions, even recursive definitions, can be handled with only minor extensions to the framework.

This is all quite easy and conforms to a common thread of both theoretical and practical work. It also matches how StarDog ICV works (as the theoretical underpinning of StarDog ICV is one of these theoretical results). Further, the approach can be implemented by translation into SPARQL queries, showing that it is practical. (There may be some constructs of OWL that do not translate into SPARQL queries when working with complete information, but at least the parts of OWL that correspond to the usual recognition conditions do so translate.)

—–

This mailing list provides plenty of evidence that OWL is not expressive enough and something on the level of SPARQL is needed. ShEx has an escape mechanism to fall back to SPARQL [1], and IBM has explicitly stated that this is required too [2].

So as a minimum, the outcome of this group should be to agree on a high-level vocabulary (possibly OWL and/or Shapes) plus a mechanism to represent other cases with something like SPARQL.

7.1 Why OWL Closed World for RDF Validation?

There are some key principles that need to be right. I believe that lack of the wider use of Semantic Web standards, has to do with a small handful of assumptions that ended up being obstacles instead of enablers. Commitment to the OWA to make monotonous reasoning possible, in my view, was one of them. On one hand, this decision has many implications that are too strange for people. On the other hand, monotonous reasoning is only possible when the asserted facts never change, just get added to. In real life, this rarely happens. Data often has errors which need to be corrected, facts that were true at some point get changed, etc. With this, the pluses of OWA don't seem to compensate for its minuses.

7.2 Why NOT OWL Closed World for RDF Validation?

- OWL cannot even do basic math

- OWL cannot do string operations

- OWL does not have have a concept of "variables" that is very much needed to do complex joins.

—–

Why is OWA and validation with OWL problematic (example)?

As for an area where OWA is problematic, one need look no further than FRBRer ontology[1], which is clearly designed using OWL constraints (which I prefer to call "axioms" to avoid confusion) in a closed world way. The definitions are quite strict, with all classes disjoint each other, such that, using reasoning in the OW, any FRBRer data will be inconsistent with data not using that exact set of axioms.

We used a FRBRer dataset as a use case in the main RDFUnit paper [1] and we actually came to the same conclusions. [1] http://svn.aksw.org/papers/2014/WWW$_Databugger/public.pdf$

7.3 Discussion Why (NOT) OWL Closed World for RDF Validation?

ambiguousness: Reusing OWL syntax with a closed world interpretation is of course a seductive path but I've always been a bit uneasy about it. OWA is built in the OWL Recommendation. I would rather have a neutral language, with non-ambiguous open world interpretation in OWL, and another one in any closed-world language (SPIN, SPARQL, you name it).

Yeah, I appreciate that concern. Everyone keeps telling me that this seems like a problem in principle; apparently we're the only ones who built it *as a real thing* and *in practice* it's not a problem at all. Our customers don't find it in the least bit confusing. In fact, as we originally said, most people who wanted OWL always wanted closed world semantics anyway, so giving it to them is a big win. Would using different namespaces help in acceptance? NO to not break existing tools like Protege which knew how to manipulate that *syntax* YES avoid confusion

7.4 Unique Naming Assumption

Non-Unique Naming Assumption You can't easily use OWL to even do cardinality checking. For example, suppose you assert that :hasFather is a functional property, and you have the triples :Luke :hasFather :Anakin and :Luke :hasFather :Darth.An OWL reasoner will invoke the Non-Unique Naming Assumption to infer that :Anakin owl:sameAs :Darth.

7.5 example validation + inferencing with CWA/UNA

```
Let's take an example schema:

  my:IssueShape {
    x:submitter @my:SubmitterShape
  }
  my:SubmitterShape {
    foaf:name LITERAL
  }

We can express that as OWL CWA/UNA like so:

  x:submitter a owl:ObjectProperty .

  my:IssueType
    rdfs:subClassOf
      [ owl:onProperty x:submitter ;
        owl:cardinality 1 ] ,
      [ owl:onProperty x:submitter ;
        owl:allValuesFrom my:SubmitterType ] .

  foaf:givenName a owl:DatatypeProperty ;
    rdfs:range xsd:string .
  foaf:familyName a owl:DatatypeProperty ;
    rdfs:range xsd:string .

  my:SubmitterType
    rdfs:subClassOf
      [ owl:onProperty foaf:givenName ;
        owl:cardinality 1 ] ,
      [ owl:onProperty foaf:familyName ;
        owl:cardinality 1 ] .

A piece of instance data like

  <Issue1> x:submitter _:submitter .
  _:submitter foaf:givenName "Bob" .

has two untyped nodes. In order to validate, We posit that:

  <Isseu1> a my:IssueType .

and the system then gets to infer that:

  _:submitter a my:SubmitterType .

(Of course, this could have been an anonymous class in the OWL, but I
gave it a label for clarity here.) CWA validation leads us to believe
that the foaf:familyName property is missing for _:submitter.

If any arcs are missing from _:submitter other than a type arc, those
are considered validation errors, but in order to even validate, I
believe that OWL UNA/CWA (e.g. ICV) is required to do type inferencing
in order to get anything done. It was this juxtaposition that I was
trying to point out.
```

8 Modularize and Separate Semantics and Constraints

semantics (OWL ontologies)

In some cases, inferred triples are not intended or do not make sense within a specific application.

Constraints are application-specific. When combining terms from multiple sources, the constraints associated with a term often depend on how the term is used in a particular application. Consider DC Terms. It was designed to provide a highly reusable vocabulary. There are very few constraints imposed by DC Terms and this makes it very reusable. For example, consider dcterms:title. In one application dcterms:title may be a required property while in another it may be optional. So where do you put those constraints? They do not belong in the DC Terms vocabulary. You need another place to define constraints and associate them with the application.

Different consumers of information conforming to an ontology may want different constraints to be checked against that ontology.

Inferencing and Constraint Validation. So if you use RDFS and OWL, and include a lot of constraints, then the terms are not very reusable because they carry a lot of baggage in the form of inferences which may not make sense in your application. For example, if you used rdfs:domain or rdfs:range then you infer certain type triples. I once was looking for terms that defined start and stop dates. I found dtstart and dtend in http://www.w3.org/2002/12/cal/ical but when I looked at the RDFS/OWL it had a lot of undesired inferences concerning the domain. So if I had reused dtstart in my application, then a reasoner would have inferred a lot of triples that didn't make sense.

Reuse of Terms from Vocabularies. At OSLC we are trying to integrate information from a lot of applications. We try to reuse terms from established vocabularies. We define new common terms in a core OSLC vocabulary. We do this to make writing queries easier. This means that resources contain terms from many vocabularies.

Modularization of ontologies and constraints.

All these complaints about RDFS and OWL appear to be based on the conception that RDFS and OWL work with a single document containing everything that can ever be said about a particular vocabulary.

However, this is a misconception. It is certainly possible in RDFS and OWL to have multiple documents that speak to the same vocabulary. This is done, for example, when OWL ontologies are extended. A core document talks about the core vocabulary and other documents add new vocabulary and add new information about the core vocabulary as well.

The exact same thing can (and should) be done with constraints. You can have a document that provides the core information about a vocabulary. You can have multiple documents that provide constraints on this vocabulary for various purposes. When validating some information one can pick which set or sets of constraints to apply.

This division can also be done with the axioms of the ontology. There is no need for all the axioms of an ontology to be in the same document, leading to the possibility of having the RDFS portion be in one document and the non-RDFS portion in another document. Other divisions are also possible.

[Assume w.l.o.g. that we are only considering standardized OWL semantics.*] If an axiom is part of the core meaning of a term then it belongs in the core ontology. For example, if it is part of the definition of the term 'bicycle' in the ontology that it must have exactly two wheels, this axiom belongs in the core ontology. Unicycles are thus excluded.

Bicycles with 'training wheels' attached may or may not qualify, based on how 'training wheels' are defined. Removing a core axiom from the ontology does not provide more opportunities for *reuse* ; it provides more opportunities for *misuse*. If an axiom is not part of the core meaning of a term, then it does not belong in the core ontology. If there are no meaningful cardinality constraints on the number of titles a thing can have then they don't belong in the core ontology. Fortunately it is possible to build new, more restrictive ontologies that reuse and restrict existing ontologies. If one wishes to describe a world where bicycles can only have wheels with spokes (e.g. things with solid wheels are not within the meaning of 'bicycle') , one can use an import statement to include the axioms from another ontology, then add the additional axioms that commitments only in this new ontology.

If you are engaged in an information fusion task, the more precise the definitions used in the shared ontology, the more feasible the task becomes. The more diverse the information contexts become the less useful generic terms become. Having a broad property named "health" is useless if in one part of the organization it refers to having up to date vaccinations , and in another it refers to the whether an aircraft is ready to deploy.

separate declarations (shapes, constraints) and OWL restrictions (semantics). The declarations of classes, properties and instances can happen in one graph, and then there can be multiple interpretations, including specifications in multiple languages. OWL already has owl:imports, SPIN already has spin:imports, ICV has ic:imports. RDF should probably have rdf:imports (and rdf:Graph similar to owl:Ontology).

I guess the question remains is what to do with the published vocabularies that do not honor this separation and mix the OWL restrictions and class definitions into a single file. Even rdfs:domain and range statements are often too restrictive. One answer is to simply ignore those statements and start with the languages in the right hand side of the diagram. After all, an rdfs:domain statement is just another triple. But there will be applications where mixing Shapes or SPIN with OWL semantics is desirable. There is no single truth here, and the application developer is already allowed to choose.

—

Our suggestion: Separate facts, axioms, and constraints

- documents containing facts (import other documents) - documents containing OWL axioms / semantics / ontology - documents containing constraints

8.1 difference between ontologies and constraints

To make an analogy with natural language, we have words and they are defined in a dictionary. We also have grammars and they define how the words are combined. We have the general rules of grammar for ordinary prose. We also have other types of grammars for specialized texts such as poetry. There are rules for sonnets, limericks, etc. The dictionary is analogous to the RDFS or OWL document. The grammar is analogous to a Shape document.

8.2 vocabularies/ontologies and constraints are independent

create / maintain / control vocabularies/ontologies and constraints independently

They (and I) want one party to create and maintain a vocabulary/ontology, while multiple other (unrelated) parties are able to create and maintain reusable sets of constraints involving that vocabulary (and not violating the constraints in that ontology).

difference between ontology and vocabulary

They presume something which perhaps you do not, which is that the party which hosts the IRIs comprising a vocabulary is empowered to maintain the definitive ontology for that vocabulary. That's why I wrote "vocabulary/ontology", because with this Linked Data approach, the two go together. I think this approach is widely accepted, although I don't think it's explicit in any W3C Recommendation.

What I hear you saying here is that regardless of that, it's reasonable for other people to publish/maintain what we might call a "secondary" or "third-party" ontology, which contains axioms concerning someone else's vocabulary. My sense is this kind of thing is generally regarded as a Bad Practice, although I don't offhand know why, and there might not be any good reason. I think I've seen people yelled at for suggesting something like this (which doesn't necessarily mean it's a bad idea).

For my use case, I think the important thing is to be able to attach constraints to interfaces. To simplify somewhat, I want to say when you're POSTing to URL U, if you want things to work properly, you MUST include exactly one foaf:firstName on each instance of foaf:Person.

I can imagine writing that as:

¡U¿ eg:interfaceConstraints ¡UC¿

and then ¡UC¿ would contain OWL of the form:

foaf:Person owl:subClassOf [a owl:Restriction; owl:cardinality 1; owl:onProperty foaf:firstName].

I don't really know Manchester syntax, but maybe it would be something like:

Class: foaf:Person Types: foaf:firstName exactly 1

Is that how you would propose I get the functionality I want?

Somewhere I'd need to say that this OWL is to be understood with closed world semantics, right?

Are there other problems I might have?

—–

axioms restricting base ontologies

It is absolutely normal to assert axioms in an ontology that restrict the models of terms in another ontology. It's an important part of ontology modularity. Think about all the axioms that are defined on Thing and Nothing. The key things to realize is that these axioms *only hold for that ontology*, and that statements made that are true in the restricted ontology are true in the unrestricted ontology, but not vice versa. *ANY* method you use to constrain the set of things that some ontology allows to exist is *BY DEFINITION* creating a new ontology using the terms of the original.

constraints and vocabularies/ontologies are dependent

One can argue that particular vocabularies/ontologies should permit multiple sets of constraints. I agree with this argument. This does not make the constraints independent of the vocabulary or ontology, however. In fact, every example of ShEx that I have seen is very tied to a particular vocabulary.

use of OWL for validation:
(1) import base ontology + add. axioms
(2) import set of constraints (for different uses)

OWL and RDFS do not fail on this point at all. One can use OWL and RDFS in a very flexible manner, where there is a base ontology and additional axioms. A constraint system using OWL or RDFS can work in a similar fashion.

Even StarDog ICV can be used in this manner. All you have to do is have an overall file that imports the base ontology and separately constraint-imports the constraints. Different uses can have the same ontology and different constraints. Some uses can even have just the constraints. One could also have a trivial modification of StarDog ICV that had an extra explicit input - the constraints.

9 constraints vs. axioms

see R-201-SEPARATION-OF-CONSTRAINTS-AND-ONTOLOGY-SEMANTICS

10 separate constraints and axioms

10.1 disjointness

A disjointness constraint is actually different (both semantically and in practice) from the OWL disjointness axiom. I see no problem with adding disjointness to an AP where such disjointness is needed in the applications that will use the AP (data creation, quality control, documentation of your "data set"). In fact, I believe I have seen such constraints being used in validation functions that use SPARQL, and it doesn't seem to be a problem. As we know (and Tom and I will show some examples in Austin), disjointness in the open world can be ill-advised precisely because it does not mean the same as disjointness in the closed world. In the closed world it may be a common requirement; in the open world it can greatly interfere with data compatibility. The same is true of cardinality (which has a very different meaning in the open world).

I do not see APs as duplicating the OWL axioms – although the natural language terms we use may be the same, the functionality is quite different.

Obviously, this is what we need to keep in mind as we go through our use cases. I don't know what problems might arise when we get into the details.

11 Diskussionen

Man müsste mit der Behauptung "vollständig gegen OWL 2 Konstrukte" vorsichtig sein, in einer gewissen Lesart von Vollständigkeit bist du vielleicht nicht vollstädnig. Überprüfts du auch so Sachen, die sich durch folgende Konmbination an Axiomen und Assertions ergeben? Hier ein Beispiel.

(1) $\exists teachesStudent.\top \sqsubseteq TeacherUniversity$
(2) TeacherUniversity $\sqsubseteq EmployeeUniversity$
(3) EmployeeUniversity $\sqsubseteq \neg EmployeeCompany$
(4) teachesStudent(eckert,schmidt)
(5) EmployeeCompany(eckert)
(1)+4) = (6) TeacherUniversity(eckert)
(6) + (2) = (7) EmployeeUniversity(eckert)
(7) + (3) = (8) $\neg EmployeeCompany(eckert)$

(8) und (5) sind im direkten Widerspruch. Das Problem ist hier dass Reasoning und Constraints sozusagen zusammenarbeiten, und mir ist kein Ansatz bekannt, der das Reasoning für OWL 2 in Query Rewriting erledigen kann.

Aber vielleicht wird das Beispiel ja tatsächlich von deinem Ansatz erwischt (tatsächlichkann man etwas definieren, was für dieses Beispiel vollständig ist = etwas was für DL lite vollständig ist, wenn ich grade nix falsch sehe).

12 Validation in RDF, XML, and Relational Worlds

12.1 Similarities

12.2 Differences

Difference between XML world and RDF world XML document with correct syntax-¿Well-formed

XML document with correct syntax and accomplishing with a DTD—XML Schema-¿Well-formed and Valid

check if a RDF graph is ”well formed” (RDF syntax, etc.)

check if the RDF graph is *valid* under a certain taste RDFS, OWLx, etc. (integrity and cardinality constraints, etc with a reasoner, SPIN, etc.) (for instance cardinality of properties and data types)

13 Related Work

- See e.g. chapter 6 of Jiao Tao's dissertation - http://tw.rpi.edu/web/doc/JiaoTaoDissertation .

CWA.

alternate semantics for OWL (using CWA):

Several years ago Evren Siren et al proposed an alternate semantics for OWL (using CWA) so that it could be used for integrity constraint (IC) checking and this is implemented in the Stardog database.

However, this OWL IC semantics was never submitted to a standards organization.

14 Conclusion and Future Work

References