OSLC Resource Shape
A language for defining constraints on Linked Data

Arthur G. Ryman
IBM Rational
DE, Chief Architect, Reporting & Portfolio and Strategy Management
+1 (905) 413-3077
ryman@ca.ibm.com

Arnaud J Le Hors
IBM
Software Standards Architect
+1 (720) 396-5228
lehors@us.ibm.com

Steve Speicher
IBM
STSM
OSLC Lead Architect
+1 (919) 254-0645
sspeiche@us.ibm.com

ABSTRACT

IBM has for several years been employing a read/write usage of Linked Data as an architectural style for integrating a suite of applications. [1]

We are encouraged by the work done by the W3C Linked Data Platform Working Group which is chartered to produce a W3C Recommendation for HTTP-based (RESTful) application integration patterns using read/write Linked Data .

The Linked Data Platform Recommendation will provide the industry with a solid foundation to build on. Yet, more work will need to be done to address in a standard way the needs of enterprise solutions that use Linked Data as an application integration platform. One such need is a type definition language that can be used to communicate and validate constraints on RDF data.

This paper explains the need for such a language, why standards like RDFS and OWL are not suitable answers and, finally, introduces OSLC Resource Shapes as a proposed solution.

General Terms

Management, Design, Standardization

Keywords

Linked Data, Type Definition, Integrity Constraints, Validation, Application Integration, Standards

1. INTRODUCTION

The W3C Linked Data Platform (LDP) Working Group (WG) [2] is chartered to produce a specification which builds on Tim Berners-Lee's 4 rules [3] and defines a standard way of manipulating RDF resources over HTTP [4] in a RESTful manner. [5]

The LDP specification [6] defines several additional rules LDP client and servers must comply with. The specification describes how each HTTP verb is to be handled - what is to be submitted by the client, what the server must do, and what the client is to expect as a result.

The LDP specification introduces the notion of LDP Resource with additional constraints over what RDF [7] requires to increase interoperability. For instance, LDP requires a resource type to be set explicitly.

However, the LDP specification falls short of defining how applications that build on LDP are to find the constraints that govern these resource types – how an LDP client might discover which properties are required on a given type and how an LDP server might validate content submitted by a client.

W3C provides several standards such as RDFS [8] and OWL [9] to describe vocabularies and ontologies in RDF but these techniques are not suitable to the problem at hand. Indeed, these standards are primarily designed to support reconciliation of different vocabularies to facilitate integration of various data sets and reasoning engines which have the ability to infer new information from given information.

Unfortunately, as we will demonstrate, although powerful, this ability means that reasoning engines function in a way that is actually contrary to what is necessary to enable the type of validation robust applications development requires.

For that reason, IBM developed as part of the Open Services for Lifecycle Collaboration (OSLC) initiative [10] a technique called Resource Shape [11] which we will briefly present in this paper. This technique consists of an RDF vocabulary that can be used for specifying and validating constraints on RDF graphs. Resource Shapes provide a way for servers to programmatically communicate with clients the types of resources they handle and to validate the content they receive from clients.

In some sense Resource Shapes do what naive users expect of RDFS and OWL.

2. RELATED WORK

There is surprisingly little literature to be found on the subject of RDF validation and language constraints for RDF. Notable exceptions include Jiao Tao's *Adding Integrity Constraints to the Semantic Web for Instance Data Evaluation* proposal [12] which provides for good background on the topic and refers to what is being discussed here as "integrity constraint" validation. However, the paper proposes to address the need for integrity constraints validation by reusing OWL with a different semantics. *Validating RDF with OWL Integrity Constraints* from Clark & Parsia, LLC [13] builds on the same idea.

While there is certainly an appeal to reusing existing technology, using the same syntax with two different semantics isn't without disadvantages. So, instead, the proposal discussed here chooses a path that stays clear of OWL which was designed for a different

LDOW2013, May 14, 2013, Rio de Janeiro, Brazil.

purpose. Other approaches such as that based on the use of a rule engine like SPIN [14] are also worth considering.

3. THE NEED FOR A CONSTRAINT LANGUAGE

Linked Data fuses REST and RDF by requiring that resources should be identified with dereferenceable HTTP URIs and that HTTP clients should be able to get RDF representations of resources.

LDP takes this concept further and defines a broadly applicable RESTful RDF based platform. With this platform developers will be able to build applications by integrating different components that function as REST services exchanging data in RDF.

RDF has the happy characteristic that "it can say anything about anything." This means that, in principle, any RDF resource can have any property and there is no requirement that any two resources have the same set of properties, even if they have the same type or types.

In practice, though, the properties that are set on resources usually follow regular patterns that are dictated by the uses of those resources. Although a particular resource might have arbitrary properties, when viewed from the perspective of a particular application or use case, the set of properties and property values that are appropriate for that resource in that application will often be predictable and constrained.

In this context, it is natural for developers to expect to be able to define the constraints governing the RDF resources they use in their application and to be able to validate against those constraints the content sent by clients to servers.

Defining the content of RDF payloads (HTTP request or response) is part of the REST service interface.

It is sound engineering practice to define interfaces between components in a system. The interface definition defines the contract between the provider and consumer of a component. For software systems, the main part of the interface definition is a precise specification of the inputs and outputs.

Type definition languages are used for this purpose, both to programmatically communicate the data an application can receive and to validate the data it receives.

LDP resources are represented as RDF graphs around which REST service interfaces are defined. A type definition language for LDP would therefore let us describe RDF graphs. Such a description would help consumers and providers determine if a given graph satisfies the REST interface contract.

Consider a simple Web application that hosts resources about change requests. We'll use the class oslc_cm:ChangeRequest to define the class of change requests. Assume there is a REST service where we can POST HTTP requests to create new oslc_cm:ChangeRequest resources. The REST service looks at the HTTP request, and if it contains an oslc_cm:ChangeRequest resource, it will create a new resource and copy the properties from the HTTP request to it. The following HTTP POST request body should succeed:

Example 1. HTTP POST changeRequest.ttl

```
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix oslc_cm: <http://open-services.net/ns/cm#> .

<http://example.com/resource>
  a oslc_cm:ChangeRequest ;
  dcterms:title "Null pointer exception in web ui" ;
  oslc_cm:status "Submitted" .
```

A type definition language would provide a way of ensuring that the resource that is submitted is of type oslc_cm:ChangeRequest and has the necessary properties.

Unfortunately there is currently no such type definition language for RDF.

4. WHY RDFS AND OWL ARE NOT SUITED FOR THE TASK

RDF Schema (RDFS) is a language for describing vocabularies and is often misconstrued as being to RDF what XML Schemas [15] are to XML. Despite the similar names these two technologies serve two very different roles. While XML Schemas are well suited to validate inputs, RDFS is not.

RDFS defines the classes rdfs:Class and rdf:Property which are used to classify terms as either classes or predicates. This limited subset of RDFS constitutes a very simple type definition language.

However, RDFS also contains other terms, such as rdfs:domain, rdfs:range, rdfs:subClassOf, and rdfs:subPropertyOf, which go beyond mere vocabulary definition and enter into the world of ontologies. The primary difference between a vocabulary and an ontology is that an ontology includes inference rules which let you infer new information from given information. This is where RDFS and OWL, which provides augmented capabilities, diverge from traditional type definition languages such as XML Schemas. Technically, the inferences are computed by a software component called a reasoner.

The function of a reasoner is very different from that of a validator and trying to use a reasoner as a validator can prove to be a very frustrating exercise.

Considering our example of a Web application handling change requests, the designer of the service could declare the domain of the oslc_cm:status property to be oslc_cm:ChangeRequest using the following RDFS statement:

oslc_cm:status rdfs:domain oslc_cm:ChangeRequest .

However, the semantics of the rdfs:domain assertion is not a constraint that says you can only use oslc_cm:status on oslc_cm:ChangeRequest resources. Rather, it is an inference rule that says if you use oslc_cm:status as a property on any resource, then that resource is classified as an oslc_cm:ChangeRequest. More precisely, the meaning of this statement is that if any statement uses the predicate oslc_cm:status then we can infer that the subject of the statement is a member of the class oslc_cm:ChangeRequest.

Similarly to rdfs:domain, RDFS also defines the predicate rdfs:range which lets us infer the class membership of the object of any statement that uses a given predicate.

Consider the following HTTP POST request, where the explicit triple stating that the resource is an oslc_cm:ChangeRequest has been omitted:

Example 2. HTTP POST changeRequest-implicit.ttl

```
@prefix dcterms: <http://purl.org/dc/terms/> .

@prefix oslc_cm: <http://open-services.net/ns/cm#> .

<http://example.com/resource>
  dcterms:title “Null pointer exception in web ui” ;
  oslc_cm:status “Submitted” .
```

From the traditional viewpoint, this HTTP POST request should fail because the server can’t find an oslc_cm:ChangeRequest resource. However, from the ontology viewpoint, it should succeed because of the semantics of RDFS.

An RDFS reasoner would infer from the explicit triples in the HTTP POST request and the service ontology that the HTTP POST request implied a triple stating that the resource was an oslc_cm:ChangeRequest.

RDFS contains several other terms, e.g. rdfs:subClassOf, rdfs:subPropertyOf, that look like common type definition language constraints, but are in fact inference rules. OWL also looks like a type definition language but in fact greatly expands on the set of inference rules and is equally unsuited to validating inputs to REST services.

OWL is so much more expressive than RDFS that it is possible for an OWL reasoner to infer mutually contradictory triples from a given graph, in which case the graph is said to be inconsistent. This ability looks, at first glance, like a potentially useful constraint checking mechanism. Unfortunately, an OWL reasoner will go to great lengths to make some superficially inconsistent-looking graphs consistent.

For example, consider the following ontology:

Example 3. OWL Ontology hasOwner.ttl

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

@prefix ex: <http://example.org/ns#> .

ex: a owl:Ontology .

ex:ChangeRequest a owl:Class ;
  rdfs:isDefinedBy ex: .

ex:Owner a owl:Class ;
  rdfs:isDefinedBy ex: .

ex:hasOwner a owl:ObjectProperty,
  owl:FunctionalProperty ;
  rdfs:isDefinedBy ex: .

ex:Joe a ex:Owner .

ex:Bob a ex:Owner .

ex:MyRequest a ex:ChangeRequest ;
  ex:hasOwner ex:Joe, ex:Bob .
```

This ontology defines the classes ex:ChangeRequest and ex:Owner and the property ex:hasOwner. This property is asserted to be a functional property, which means that it is single-valued, i.e. for any given subject there must be at most one object. The ontology also describes two owners, ex:Joe and ex:Bob, as well as a change request, ex:MyRequest, and asserts that this change request has two owners, ex:Joe and ex:Bob. This looks like a contradiction. It would be nice if a type checker could flag this.

An OWL reasoner will not say that this ontology is inconsistent because OWL does not make the “Unique Name Assumption”. This is a fundamental aspect of Web architecture [16] since there is no requirement that every resource have a unique URI. In fact, it is common for synonyms to be defined in different vocabularies. Given the above ontology, an OWL reasoner will find no inconsistency.

An OWL reasoner will judge an ontology to be consistent if there is some world in which the ontology makes sense. In this case, the ontology makes sense when ex:Joe and ex:Bob identify the same resource. The ontology is said to entail this implication. OWL has the property owl:sameAs which asserts that its subject and object identify the same resource. Thus the following triple is entailed by the ontology.:

ex:Joe owl:sameAs ex:Bob .

Although logical, this entailment makes reasoners unsuitable to the task of validating RDF content sent to an LDP server.

5. OSLC RESOURCE SHAPES

Linked Data programmers have a legitimate need to be able to specify constraints on data, e.g. as preconditions in REST APIs. OO programmers are used to specifying constraints on data with a variety of traditional type definition languages such as Java, UML, and XML Schema. As previously discussed RDFS and OWL are very different from traditional type definition languages and are therefore not the solution. The OSLC Resource Shape specification is a proposed solution for specifying constraints on RDF data.

A resource shape is a set of grammar rules, expressed in RDF, an RDF graph must comply with to be correct. A resource shape lists the properties that are expected or required in a graph, their occurrence, range, allowed values, etc.

A resource shape lets you determine if a given graph is valid or invalid. A resource shape checker could be implemented as a set of SPARQL ASK queries [17] on the graph. A SPARQL ASK query is a query whose result is either true or false. If all the SPARQL ASK queries return true then the graph is valid, otherwise it is invalid.

To briefly illustrate shapes, suppose that in our oslc_cm:ChangeRequest example we require that when a new resource is created, it must have exactly one dcterms:title property and zero or one oslc_cm:status property. These constraints are expressed in the following simplified resource shape:

Example 4. OSLC Resource Shape changeRequest-shape.ttl

```
@prefix dcterms: <http://purl.org/dc/terms/> .

@prefix oslc: <http://open-services.net/ns/core#> .

@prefix oslc_cm: <http://open-services.net/ns/cm#> .

@base <http://example.com/shape/oslc-change-request> .
```

```
<> a oslc:ResourceShape ;
  dcterms:title "Creation shape of OSLC Change Request" ;
  oslc:describes oslc_cm:ChangeRequest ;
  oslc:property <#dcterms-title>, <#oslc_cm-status> .

<#dcterms-title> a oslc:Property ;
  oslc:propertyDefinition dcterms:title ;
  oslc:occurs oslc:Exactly-one .

<#oslc_cm-status> a oslc:Property ;
  oslc:propertyDefinition oslc_cm:status ;
  oslc:occurs oslc:Zero-or-one .
```

This resource shape specifies constraints governing an oslc_cm:ChangeRequest resource. It uses the property oslc:occurs to specify the occurrence constraints of the dcterms:title and oslc_cm:status properties. Specifying the occurence of a property as either oslc:Exactly-one or oslc:Zero-or-one constrains the property to be functional, which is what we were trying to achieve through the use of owl:FunctionalProperty in Example 3. OWL Ontology hasOwner.ttl.

As mentioned above, each constraint can be expressed as a SPARQL ASK query. For example, the following query checks the occurrence of the oslc_cm:status property:

Example 5. SPARQL Query ask-oslc_cm-status-occurs.rq

```
prefix oslc_cm: <http://open-services.net/ns/cm#>

ask {
    select ?resource
    where {
      ?resource a oslc_cm:ChangeRequest.
      ?resource oslc_cm:status ?status
    }
    group by ?resource
    having (count(?status) <= 1)
}
```

This query uses SPARQL aggregation to count the occurrence of the oslc_cm:status property and compare it to the constraint specified in the shape document.

Running this query on the HTTP POST body in Example 1. HTTP POST changeRequest.ttl returns true. This result confirms that the shape is valid with respect to this occurrence constraint.

For a counter-example, consider the following HTTP POST which has two values for the oslc_cm:status property:

Example 6. HTTP POST changeRequest-2.ttl

```
@prefix dcterms: <http://purl.org/dc/terms/> .

@prefix oslc_cm: <http://open-services.net/ns/cm#> .

<http://example.com/resource> a oslc_cm:ChangeRequest ;
  dcterms:title "Null pointer exception in web ui" ;
  oslc_cm:status "Submitted", "Working" .
```

Running the same query returns false, because oslc_cm:status occurs twice.

OSLC Resource Shapes let you express many other common constraints in addition to occurrence constraints.

A Resource Shape lists the properties that are allowed or required for a specific type of resource. For each property, it specifies the type of its value, the number of times it is expected to occur, and whether it is required. A default value as well as a list of possible values can be provided. In addition, for properties for which the value is a resource, a shape can be provided for that resource, allowing for a recursive model.

The following table lists some of the property constraints that can be specified. See *OSLC 2.0 Appendix A: Common Properties* [11] for the complete specification.

Name	Description
valueType	The type of value the property can have. This can be one of the following: Literal value-types: • Boolean • DateTime • Decimal • Double • Float • Integer • String • XMLLiteral Resource value-types: • Resource • Local Resource • AnyResource When omitted, the value type is unconstrained.
range	When valueType is a resource value-type, this can be used to specify the resource type allowed. The default is Any.
valueShape	When valueType is a resource value-type, this can be used to specify the Resource Shape for the value. Note that this allows various shapes to be associated with the same type.
allowedValues	Specifies an oslc:AllowedValues resource which lists the allowed values for the property.
allowedValue	A value allowed for the property. If there are both allowedValue elements and an allowedValue resource, then the full-set of allowed values is the union of both.
defaultValue	A default value for the property.
maxSize	For String properties only, this specifies as an integer the maximum number of characters allowed. If not set, then there is no maximum or maximum is specified elsewhere.
occurs	Either Exactly-one (the property is required), Zero-or-one (the property is optional), Zero-or-many (the property is optional), or One-or-many (the property is required)

Name	Description
readOnly	A Boolean specifying whether the property is read-only. If omitted, or set to false, then the property is writable.

Although implementations of the specification are not required to use SPARQL to check constraints the meaning of each constraint can be expressed in terms of a suitable SPARQL ASK query in a way similar to what we showed in Example 5. SPARQL Query ask-oslc_cm-status-occurs.rq.

As part of the OSLC initiative various Resource Shapes have been developed and successfully used in different application domains including Application Lifecycle Management (ALM) and Integrated Service Management (ISM) to describe resources such as a Change Request [18], a Test Case [19], a Requirement [20], or a Performance Monitoring Record [21]. We have found this technique to adequately address the need for describing the data that application specific Linked Data services expect, and for these services to validate the data they received from clients.

6. CONCLUSION

Linked Data fuses REST with RDF. Sound software engineering practices dictate that we clearly specify REST interfaces. Traditional approaches, such as XML Schema, don't apply to RDF, and RDF ontology languages such as RDFS and OWL are not suitable to the task. We therefore need an RDF-friendly way to describe Linked Data REST interfaces that we will be able to use with LDP. Based on our experience in OSLC, we believe Resources Shapes are a possible solution to this need but more importantly we believe the industry needs a standard solution to this problem.

7. ACKNOWLEDGMENTS

This paper contains material and concepts that come from our work in OSLC and Arthur Ryman's *Linked Data Interfaces* article [22].

8. REFERENCES

[1] Arnaud J Le Hors and al. *Using read/write Linked Data for Application Integration.* LDOW2012, April 16, 2012. http://events.linkeddata.org/ldow2012/papers/ldow2012-paper-04.pdf

[2] W3C Linked Data Platform Working Group http://www.w3.org/2012/ldp

[3] Tim Berners-Lee. *Linked Data Design Issues.* 2006 http://www.w3.org/DesignIssues/LinkedData.html

[4] R. Fielding and al. *Hyper-text Transfer Protocol (HTTP/1.1)*, IETF RFC2616, 1999. http://tools.ietf.org/html/rfc2616

[5] Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures.* Doctoral dissertation, University of California, Irvine, 2000.

[6] Steve Speicher, John Arwe. *Linked Data Platform 1.0.* W3C, 2012. http://www.w3.org/2012/ldp/hg/ldp.html

[7] Graham Klyne, Jeremy J. Carroll. *Resource Description Framework (RDF).*W3C, 2004 http://www.w3.org/TR/rdf-concepts/

[8] Dan Brickley and al. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C, 2004 http://www.w3.org/TR/rdf-schema/

[9] W3C OWL Working Group. *OWL2 Web Ontology Language Overview.* W3C, 2012 http://www.w3.org/TR/owl2-overview/

[10] *Open Services for Lifecycle Collaboration (OSLC)* http://open-services.net

[11] Dave Johnson, *OSLC 2.0 Appendix A: Common Properties.* OSLC, 2012 http://open-services.net/bin/view/Main/OSLCCoreSpecAppendixA

[12] Jiao Tao, *Adding Integrity Constraints to the Semantic Web for Instance Data Evaluation*, in Proceedings of the 9th International Semantic Web Conference (ISWC 2010), Shanghai, China, November 7-11, 2010. http://www.cs.rpi.edu/%7Etaoj2/2010/iswc2010dc.pdf

[13] Héctor Pérez-Urbina an al. *Validating RDF with OWL Integrity Constraints.*Clark & Parsia, LLC. 2010-2012. http://stardog.com/docs/sdp/icv-specification.html

[14] Holger Knublauch and al, *SPIN Member Submission*, W3C, 2011. http://www.w3.org/Submission/spin-overview/

[15] Paul Biron, Ashok Malhotra. *XML Schema Part 2: Datatypes,* Second Edition, W3C, 2004 http://www.w3.org/TR/xmlschema-2/

[16] Ian Jacobs, Norman Walsh. *Architecture of the World Wide Web*, W3C. 2004. http://www.w3.org/TR/webarch/

[17] Lee Feigenbaum and al. *SPARQL 1.1 Protocol*, W3C, 2013 http://www.w3.org/TR/sparql11-protocol/

[18] Steve Speicher. *Open Services for Lifecycle Collaboration (OSLC) Change.* OSLC, 2010. http://open-services.net/bin/view/Main/CmSpecificationV2

[19] Paul McMahan. *Open Services for Lifecycle Collaboration (OSLC) Quality Management Version 2.0*. OSLC, 2011. http://open-services.net/bin/view/Main/QmSpecificationV2

[20] Ian Green. *Open Services for Lifecycle Collaboration (OSLC) Requirements Management Version 2.0,* OSLC, 2012. http://open-services.net/bin/view/Main/RmSpecificationV2

[21] Julianne Bielski, John Arwe. *Open Services for Lifecycle Collaboration (OSLC) Performance Monitoring Version 2.0.* OSLC, 2013. http://open-services.net/wiki/performance-monitoring/OSLC-Performance-Monitoring-Specification-Version-2.0/

[22] Arthur Ryman. *Linked Data Interfaces.* developerWorks, 19 March 2013 http://www.ibm.com/developerworks/rational/library/linked-data-oslc-resource-shapes/index.html