Using SPARQL to Validate Open Annotation RDF Graphs

A Position Paper Submitted to the
RDF Validation Workshop: Practical Assurances for Quality RDF Data
10-11 September 2013

Anna Gerber (*agerber@itee.uq.edu.au*)[1]
Timothy W. Cole (*t-cole3@illinois.edu*)[2]
David Lowery (lowery@cs.umb.edu)[3]

Community Draft 2 of the Open Annotation (OA) Specification[4] was released in February 2013 by the W3C Open Annotation Community Group.[5] This specification describes a RDF-based data model and a core ontology[6] for describing annotations of Web Resources. The OA Specification is designed to facilitate the sharing of annotations and to facilitate the emergence of a Web and Resource-centric interoperable annotation environment that allows leveraging annotations across the boundaries of annotation clients, annotation servers, and content collections. The OA Ontology is modest in scope and in numbers of classes (19), properties (23) and explicitly defined concepts (12 Named Individuals). Existing classes and properties are used (imported) where possible; in particular the OA ontology imports RDFS, the SKOS core, and W3C PROV. The OA Specification and associated OA Ontology mandate[7] that all compliant annotation RDF instances include several classes and properties. Other classes and properties are recommended, and still others are optional. As illustrated further below, the specification requires that some optional classes and properties be used in concert -- for example, the use of the SpecificResource class requires the use of the hasSource property. Cardinality constraints are also imposed -- for example, a given SpecificResource instance in an annotation graph must appear as the subject of exactly one single hasSource predicate.

Since the release of the OA Specification in February, members of the OA Community Group, led by Anna Gerber and Jane Hunter at the University of Queensland, have been exploring and experimenting with ways to validate conformance to the OA Specification. We began by reviewing and refining validation goals and requirements. We wanted a reasonably lightweight approach that was modular, leveraged existing tools where possible, and reported results in a manner that would facilitate learning about OA requirements and best practices. In other words, given the newness of the OA Specification, we assumed that implementers would appreciate validation results that reported each instance of a missing or incorrectly used class or property,

[1] ITEE eResearch Group, The University of Queensland
[2] University Library and Graduate School of Library & Information Science, University of Illinois at UC
[3] Harvard University Herbaria and Computer Science Department, University of Massachusetts at Boston
[4] http://www.openannotation.org/spec/core/
[5] http://www.w3.org/community/openannotation/
[6] http://www.w3.org/ns/oa# , http://www.w3.org/ns/oa.rdf , http://www.w3.org/ns/oa.ttl
[7] The terminology used follows RFC 2119 (http://www.ietf.org/rfc/rfc2119).

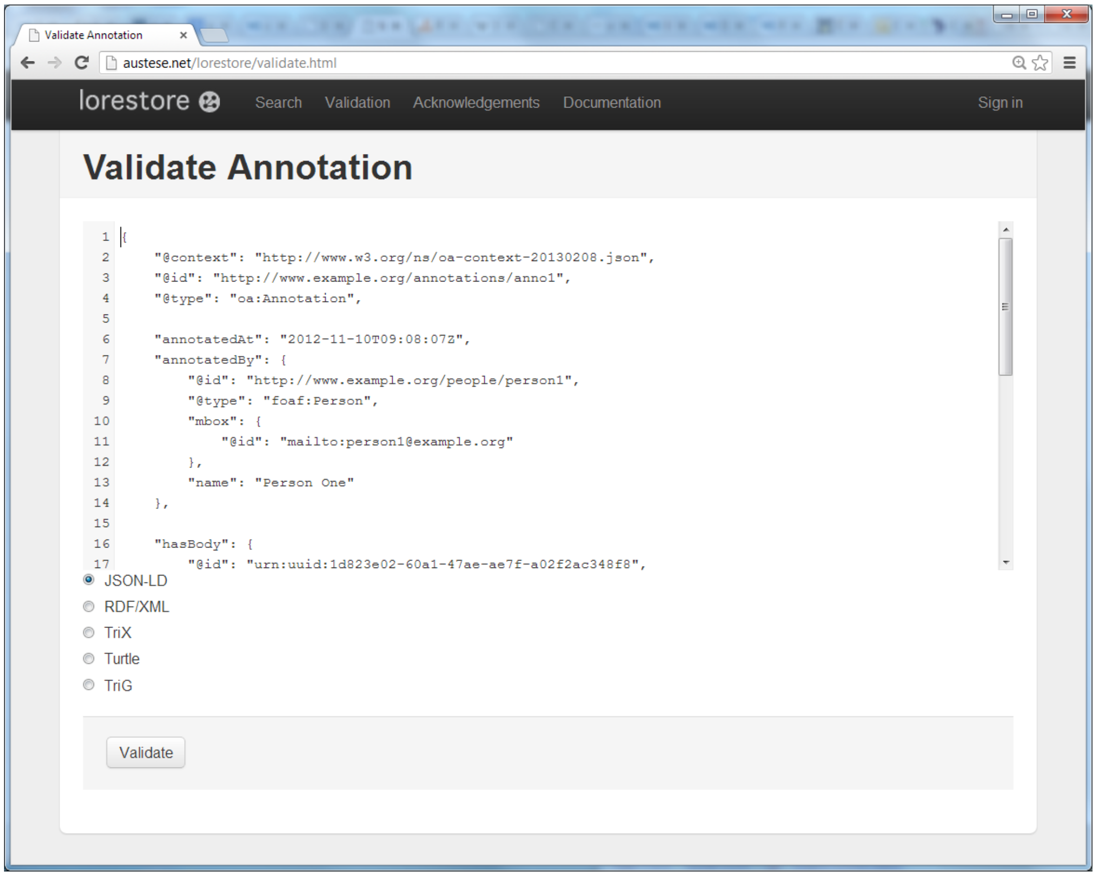
directly referencing the relevant section of the OA Specification. In practical terms we wanted to isolate and capture the rules of the OA Specification and Ontology and be able to test for compliance against each rule using an architecture that would make it easy for other developers to extend our validation tool with their own community-based rules for annotation and supplemental annotation description ontologies. With these goals in mind, cognizant of the modest size and scope of the OA Ontology, and looking at several successful implementations in other domains, we settled early on a SPARQL-based validation approach.

In pursuing a rules / SPARQL-based approach to validation we found both a number of other projects to inspire us and a foundation of tools on which we could design and build our SPARQL-based OA validator. In particular we were influenced by tools emerging from software development contexts, like Cucumber[8] and the VIATRA2 / EMF IncQuery work on ontology validation by incremental model query techniques.[9] We also looked closely at a broad range of validation systems and tools, both old and new, such as Schemarama (2001)[10] and the more recent sparql-check[11] (reworking Schemarama 2), Stardog ICV (validating RDF with OWL Integrity Constraints),[12] the W3C SPIN modeling vocabulary[13] supporting the use of SPARQL to specify rules and logic constraints, SPARQL for SKOS integrity constraints[14] and Pellet.[15] And we found several papers in the literature informative for our needs, notably an early paper by Viho Raatikka and Eero Hyvönen[16] and two papers by Evren Sirin, Jiao Tao, et al.[17] One software library that our implementation made direct use of was the rdf2go software library.[18]

Implementation -- the Lorestore OA Repository and Validator

Building on this prior art and on our design objectives, we have implemented a SPARQL-based OA validation service[19] that can be used remotely or installed locally. This OA validation service is implemented in Java and runs as a component of the UQ ITEE eResearch Lorestore annotation repository service. The source code for Lorestore, including the OA validation service, is available from GitHub.[20] The Lorestore OA validator is available as a Web Service. The tool can also be run interactively via an HTML Web form (Figure 1). An RDF instance can be posted to the service (in any of several serializations -- JSON-LD, RDF/XML, TriX, Turtle, TriG), with validation results returned in JSON.

[8] http://cukes.info/
[9] http://incquery.net//publications/trainbenchmark
[10] http://swordfish.rdfweb.org/discovery/2001/01/schemarama/
[11] https://github.com/ldodds/sparql-check
[12] http://stardog.com/docs/sdp/ , http://stardog.com/docs/sdp/icv-specification.html
[13] http://www.w3.org/Submission/spin-modeling/
[14] http://www.proxml.be/users/paul/weblog/40127/SPARQL_for_SKOS_integrity_constraints.html
[15] http://clarkparsia.com/pellet/icv/ ,
[16] http://www.seco.tkk.fi/publications/2002/raatikka-hyvonen-ontology-based-semantic-metadata-validation-2002.pdf
[17] http://clarkparsia.com/files/pdf/ic-owled09.pdf , http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1931/2229
[18] http://semanticweb.org/wiki/RDF2Go ,
[19] http://austese.net/lorestore/validate.html
[20] https://github.com/uq-eresearch/lorestore , https://github.com/uq-eresearch/lorestore/blob/master/src/main/java/net/metadata/openannotation/lorestore/servlet/rdf2go/OAValidationHandler.java



Validate Annotation

```
{
    "@context": "http://www.w3.org/ns/oa-context-20130208.json",
    "@id": "http://www.example.org/annotations/anno1",
    "@type": "oa:Annotation",

    "annotatedAt": "2012-11-10T09:08:07Z",
    "annotatedBy": {
        "@id": "http://www.example.org/people/person1",
        "@type": "foaf:Person",
        "mbox": {
            "@id": "mailto:person1@example.org"
        },
        "name": "Person One"
    },

    "hasBody": {
        "@id": "urn:uuid:1d823e02-60a1-47ae-ae7f-a02f2ac348f8",
```

[✓ JSON-LD] RDF/XML TriX Turtle TriG

Validate

Figure 1: HTML Form interface to the OA Validation Service

A total of 55 rules have been defined representing the constraints and requirements of the OA Specification and Ontology. For each rule we have defined a SPARQL query to check compliance. Because many of the OA classes and properties are optional, not all rules are applicable to all annotation instances. In addition to the SPARQL query to test for rule compliance, a separate, pre-condition SPARQL query has also been defined for each rule. This pre-condition query, which returns a Boolean True/False result, determines if the rule is applicable (and thereby determines if the primary SPARQL query associated with the rule should be run). Each pair of SPARQL queries is linked to the relevant section of the OA Specification where the requirement or constraint being tested is articulated. A severity (either 'error' or 'warn') is also associated with each rule according to whether the OA Specification requirement or constraint is expressed using 'MUST' or 'SHOULD.' The OA Specification section number, a description of the requirement or constraint, and a pre-condition message is also associated with each rule. The pre-condition message text is used to construct a 'not applicable' message to display if a given rule is not applicable to a particular annotation instance. Along with the 'status' (i.e., pass, fail, skip), and ‘result’ (non-compliant resources in the case of validation failure) of the SPARQL-based testing of the rule, all of these elements for each rule are returned to the client or user invoking the validation check. Figure 2 illustrates a typical JSON return for the check that the annotation being validated include at least one 'hasTarget' triple.

```
{
  "ref": "2.1.0. (5) Body and Target Resources",
  "url": "http://www.openannotation.org/spec/core/core.html#BodyTarget",
  "description": "There MUST be 1 or more oa:hasTarget relationships associated
with an Annotation.",
  "severity": "error",
  "preconditionMessage": "No Annotations identified",
  "precondition":   "PREFIX   oa:   <http://www.w3.org/ns/oa#>   ASK   WHERE
{{?annotation oa:hasTarget ?t}UNION {?annotation a oa:Annotation}}",
  "query":  "PREFIX  oa:  <http://www.w3.org/ns/oa#>  SELECT  ?annotation  WHERE  {
?annotation  a  oa:Annotation  .  FILTER(NOT  EXISTS  {  ?annotation  oa:hasTarget  ?t
}) }",
  "status": "pass",
  "result": ""
}
```

Figure 2: JSON Return for a Rule (1 of 55)

A Community-Specific Implementation

The OA Specification and Ontology is designed to be extended as necessary to meet the needs of specific communities. FilteredPush(FP)[21] is a platform for deploying actionable annotations of distributed, mutable biodiversity data. Its annotations are based on OA, with a small number of extensions aimed at giving change guidance to data publishers that receive annotations describing incomplete or erroneous data. Specialized FP annotations correspond to particular domain business logic operations (insert determination, update georeference, etc). Accordingly, FP validation requirements focus on insuring that consuming and producing software agree that sufficient and appropriate RDF has been delivered to allow the consumer to take appropriate action. FP adopted a similar validation strategy to LoreStore's, but its SPARQL queries serve slightly more complex validation purposes, accordingly with slightly different structure.

Validity depends on sets of rules whose structure depends not only on OA rules, but also on the producer's expectations of the consumer's actions, couched about entities in the domain vocabularies. One example of how we are using SPARQL rules to determine whether or not an annotation is a valid instance for the purposes of FP is by defining queries for valid combinations of body type and expectation. For example, a body typed as a dwc:Identification[22] that is paired with an Expectation of Update has different requirements for validity than one paired with an Expectation of Insert. There are also invalid pairings of expectation and body that must be dealt with, such as an expectation of Solve_With_More_Data paired with a body typed as a dwc:Identification, which is not a valid annotation according to our rules as there is no business logic operation that corresponds to this combination.

Applicability pre-conditions for each rule set, akin to LoreStore’s, test applicability of the entire rule set. These could even comprise run-time validation rules, changed by mutual agreement of

[21] http://filteredpush.sourceforge.net/ , http://wiki.filteredpush.org/
[22] 'dwc:' is a reference to the Darwin Core Namespace as defined by http://rs.tdwg.org/dwc/terms/

the network nodes. The structure of the set of rule sets is expressed in XML carrying references to the SPARQL queries, and this structure is constrained by an XML-Schema. Rule sets end with an optional parse rule that takes the form of a select query. The names of the variable bindings in the SPARQL are available in the configuration of the consumer. Thus, FP validation rule sets also ensure that a parsed annotation provides the fields required by the validity tests.

Interest in the RDF Validation Workshop

The Lorestore OA validator service was implemented to facilitate and encourage adoption of the OA Specification. Our goal is to provide a tool which makes it easy to validate conformance to the OA Specification and Ontology documents. We want a tool that can be used by a developer from a Web form, but could also be used for programmatic validation, both via UQ's Web Service running on the Australian NeCTAR research cloud, and in a local context (i.e., by deploying Lorestore locally). As illustrated by the FP community-specific validation example, an important next step is making it easier to extend the baseline Lorestore OA validator to test for community-specific annotation requirements and constraints. Community-specific requirements and constraints require modifying and/or augmenting the 55 rules implemented in the baseline OA Lorestore validator. What are the ways to make that easier to do and document (while staying in sync as the baseline OA validator is updated)?

We are interested in the RDF Validation Workshop in order to learn more about trends and the current state of RDF Validation tools and services and in order to better understand how we can make the Lorestore OA validator more useful and flexible. We want to compare our assumptions and design criteria with what others are using or developing. We have found the SPARQL-based validation approach efficient for our goals -- and are happy to share why we think it a good and useful approach to RDF validation -- but we also would like to learn more about ways to make our SPARQL-based approach more flexible and about other perspectives, trade-offs and alternatives that others have identified. We anticipate that we will learn of ways we can improve the quality, correctness and completeness of our existing OA validator. For example, we are considering the option of depositing the SPARQL for our baseline 55 OA validation rules into a separate git repository on github as a way to facilitate community extension. Would this be a good strategy? Have others tried this? Git forking works well for template projects which are designed to be customized. We anticipate it could work well for validation rules too. Communities could fork the base 55 rules to modify or add domain-specific rules, and any updates or changes to the base rules made over time could be merged back into the forks from upstream. Git keeps a full revision history and makes it possible to revert or override changes, allowing full control over this process. We also anticipate learning more about alternatives to SPARQL-based validation, with an eye to potentially extending and enhancing our existing tool and/or developing additional or alternative OA validation systems.