RDF Ventures to Boldly Meet Your Most Pedestrian Needs

by Eric Prud'hommeaux and Jose Emilio Labra Gayo

Linked Data and the Charm of Weak Semantics

EDITOR'S SUMMARY

Defined in 1999 and paired with XML, the Resource Description Framework (RDF) has been cast as an RDF Schema, producing data that is well-structured but not validated, permitting certain illogical relationships. When stakeholders convened in 2014 to consider solutions to the data validation challenge, a W3C working group proposed Resource Shapes and Shape Expressions to describe the properties expected for an RDF node. Resistance rose from concerns about data and schema reuse, key principles in RDF. Ideally data types and properties are designed for broad use, but they are increasingly adopted with local restrictions for specific purposes. Resource Shapes are commonly treated as record classes, standing in for data structures but losing flexibility for later reuse. Of various solutions to the resulting tensions, the concept of record classes may be the most reasonable basis for agreement, satisfying stakeholders' objectives while allowing for variations with constraints.

KEYWORDS

RDF
data structures
document schemas
validation
records
information reuse
logic

Eric Prud'hommeaux is a member of the World Wide Web Consortium (W3C) staff at MIT. He has participated in developing Semantic Web technologies such as SPARQL and RDB2RDF in order to meet use cases in the Semantic Web in Health Care and Life Sciences Interest Group. He developed Shape Expressions in order to meet validation and transformation use cases for clinical data. He can be reached at eric<at>w3.org
Jose Emilio Labra Gayo is the main researcher of the WESO (Web Semantics Oviedo) research group and an associate professor at the University of Oviedo, Spain. He can be reached through his webpage at http://di002.edv.uniovi.es/~labra/

The Resource Description Framework (RDF) data model was defined along with an XML syntax in 1999. A class hierarchy (if Spot is a dog and all dogs are animals, then Spot is an animal) and property domains and ranges followed a year later. It is perhaps unfortunate that this model came under the name *RDF Schema* (RDFS) as it didn't offer any of the data constraints available in other schema languages like SQL's DDL or W3C XML schema. In hindsight, this development path was clearly in tension with the priorities of everyday programmers and systems architects who care primarily about creating and accessing well-structured data and, perhaps secondarily, about inference. Four years after RDFS, OWL extended the facilities provided by RDFS into an expressive ontology language that could describe the information required for instances of classes. However, once again, the language was oriented toward a healthy distributed information infrastructure and not that last mile which permits developers to confidently produce and consume data. While OWL could detect errors when one used a literal with the wrong data type, it wouldn't complain if you say that every vehicle registration has an owner's first name and last name and then fail to supply those values. OWL is designed for an open world semantics, which means that it won't conclude anything (such as signaling missing data) based on the absence of provided data. The absence of evidence is not evidence of absence.

Finally, another four years later in 2008, the RDF community assembled to deliver a query language to meet the most elementary of application needs – accessing data. The language met immediately with overwhelming

This document represents the personal opinions of the authors. It does not imply any endorsement by the W3C staff or membership.

acceptance and adoption. This ability to query led to the development of many new applications, as well as databases and libraries designed to facilitate application development. This energy led to the expansion of SPARQL into Update (analogous to SQL DDL) and HTTP. It did not, however, elegantly solve the problem of RDF data description and verification. The rest of this article describes efforts to create a standard validation language and where this work stands now.

RDF Validation Workshop

In September or 2014, around 30 companies came together to describe their validation needs and solutions. The overall message was that "it would be great to use SPARQL, but we want a higher-level language." Two communities had very similar proposals, the Dublin Core community with its Description Set Profiles and Open Services Lifecycle Collaboration (OSLC) with its Resource Shapes. The principal outcome was that IBM would make a formal W3C Submission of Resource Shapes and W3C would charter a working group on RDF validation.

Well, it wasn't exactly validation because that process was only one aspect of how these descriptions would be used. Others included generating interface forms or data. It's arguable that once you have a semantic for validation, you've conquered the rest, but that doesn't mean that someone needing something for interface generation would know to look under "V" for validation. The name *Schema* was already taken by another technology. After much seeking, we decided that *Shapes* was the best we could do.

Resource Shapes

The Resource Shapes language is an expression of a collection of properties expected to be associated with some RDF node. It includes a predicate name, a cardinality (Zero-or-one, Zero-or-more, Exactly-one, One-or-more) and either a required datatype for RDF Literals or another shape for objects that are in turn complex structures. This is all written in RDF (the specifications mandate Turtle, but in principle this could come from RDF/XML, RDFa and so forth). The specification identifies a few contexts for validation, which we will call "triggers." A trigger is a statement that asserts how some data or interface through which data may pass is connected to a shape.

After the submission of Resource Shapes and another called Shape Expressions (which aimed to provide both a human-facing language and a semantic definition of Resource Shapes), the W3C staff proposed a charter based on these specifications. This proposal met with pushback from advocates for SPIN and Stardog ICV, both deployed products providing constraints. The former is a language for implementing constraints as SPARQL queries, mostly used with RDFS classes as triggers; the latter is a reinterpretation of OWL axioms with closed world and unique name assumptions.

SPARQL Inferencing Notation

SPIN is also a W3C member submission dating back to 2011. It has received some adoption but is still primarily developed and supported by Top Quadrant. SPIN is used for validation by attaching SPARQL queries to RDF classes with the implication that every instance of that class must pass the SPARQL query (actually, get zero results). SPIN has a template mechanism for substituting terms from an RDF structure into a SPARQL query. While some SPIN constraints on class members are expressed as SPARQL queries, others are RDF graph structures, somewhat like Resource Shapes.

Stardog ICV

Stardog ICV is a technology marketed by Clark & Parsia as part of the Stardog product. It reinterprets OWL axioms as constraints on RDF graphs utilizing a closed-world semantics that permits it to conclude that data has errors by assuming that it has seen all of the information. Where OWL assumes that any two URLs may stand for the same thing unless explicitly stated (with owl:differentFrom), ICV assumes that all URLs stand for different things in the domain of discourse. The combination of these two assumptions permits conventional OWL syntax to identify nodes that are missing assertions or have too many, without an exhaustive enumeration of all of the things that are different from each other.

Data Shapes Charter

Both of these communities believe that the RDF Data Shapes Working Group (WG) should not take any document as a starting document, but should instead, selecting use cases, determine their requirements and finally decide what technology to use as a starting place. There are major differences in these approaches, which has led to differences within the WG about how to model shapes and whether they should be classes and, in fact, whether all classes should be shapes.

Resource Shapes, Shape Expressions and Description Set Profiles all treat these shapes as distinct from RDF classes. RDF type assertions, the mechanism by which instance data is attached to classes, have a special status in RDF. In one sense, they're just another triple with a very popular predicate; but all of the inference systems around RDF use type triples as the centerpiece of their representational facilities and both infer type triples as well as use type triples to infer other triples. In order to understand why, we need to dive into the principles that optimize RDF for serendipitous reuse. This reuse occurs when someone uses the types and properties designed by someone else for a new purpose, for instance using the address record from a contact database as part of the data structure in a mapping application.

Data/Schema Reuse

Most non-RDF schema languages enable one to write down attribute names, type names and relationships, cardinalities and definitions. This facility effectively allows one to pick up a schema, say a UML model or XML schema of an address, and use it with reasonable confidence for its intended purpose, be that in a music catalog, medical record or large bank transfer. The problem is that these restrictions are tuned to meet the needs of a small number of applications (frequently one) because they focus on the constraints of the business process instead of the real-world entities involved in those business processes. When we want to reuse those definitions, we end up building complex class hierarchies with lots of branches to capture all of the constraints for all of the use cases we can envision. While it's possible to meet the structural informatics requirements for a small enterprise, it becomes much harder to model the more diverse needs of a more distributed organization with evolving needs. Popular RDF ontologies foster reuse of their data structures, describing the entities and leaving out the constraints that tie them to a particular use. This approach makes them available to the broadest possible set of use cases.

SemWeb netizens tend to avoid creating redundant properties and classes, instead borrowing a few predicates from this ontology and some classes or identifiers from that one. The resulting heterogeneity results in data optimized for reuse. If some issue-tracking database talks about the user who submitted a bug, they'll likely use popular identifiers from the Friend of a Friend (FOAF) ontology or Schema.org to talk about the given name, email address and so forth, adding properties from their own ontology where required, for example, to capture a relationship between entities like the submitter to some issue-tracking system. This submitter record has all of the necessary data for the issue tracker, but people, tools and queries that just know FOAF can extract the bits they need as well.

In theory, there's nothing that says that a shape can't delineate some class; it's essentially a recipe for determining membership in a set, but the implications of validity with respect to a shape become very complicated when they interact with RDFS and OWL inferences. Another problem is that these shapes that define the structure are fundamentally different from the class definitions described above. While many classes claim to model the real world, others, let's call them *record classes*, simply capture some information for a particular purpose.

Record Classes

Attaching business process-specific constraints (often cardinalities) to types demonstrably hampers reusability, but lots of folks do it, particularly if they don't mind changing definitions when their needs evolve. This practice is of course feasible in small or authoritative deployments but hard to scale to widely reused data. One reasonable compromise is to view the schema as a description, not of real-world entities, but instead of records native to some system. While it may be foolhardy to design an ontology with the assumption that a person has only one email address, it makes

perfect sense for a software system or business process to assert that it only handles one email address per user. Without considering the duality between entities and the records that describe them, it looks counter-productive to conflate shapes and classes, but classes can stand for data structures as well. If a system then conflates, for example, users and user records, it does so at the risk of sacrificing some flexibility and that ideal state of serendipitous reuse that we all appreciate in RDF.

While the "record class" solution appears to eliminate an impasse, there remains the problem of deciding how to attach shape constraints to classes. One proposal that all classes also be shapes met with resistance on multiple fronts: it makes, for example, FOAF Person a shape, which is not what people have in mind when attaching FOAF types to data; it effectively creates a new inference mechanism with new ways to create inconsistencies; and it encourages people to conflate non-record classes with the entities they represent. The arguments on the other side are compelling as well: if there is a subsumption relationship between shapes (basically, one shape can inherit from another), we end up duplicating the logic of the RDFS subclass; if we have to invent a relationship between instances and shapes, this relationship has the appearance of being a clone of RDF's type relationship tailored for use with shapes.

A practical way to model record classes is to define a relationship, say "classShape," between a class and a shape that states that every individual in that class must conform to that shape. This definition makes the assertion that a class has invariant (universal, global, ...) constraints, a conspicuous speech act. This also provides a consistent way to query for the invariants associated with a class.

In RDF, there is nothing that prevents one from using a node for multiple purposes. In OWL, this sort of "punning" allows one to use the same identifier for disjoint concepts like classes vs. individuals. This situation would use an identifier for both a class and a shape. Even if the class and the shape were punned (that is, were the same RDF node), consistent use of the classShape relationship would facilitate simpler discovery and reduce confusion and ambiguity. One could create a reflexive subproperty of "classShape," though doing so is probably not of real value.

Discriminating Types

Above, we see the tension between folks who have been using classes to define structure and those who have developed their classes for maximal reuse. Another tension within the Shapes Working Group lies in the fact that conventional SPIN relies very heavily on type assertions, either explicitly included in the data or inferred from rules about the predicates used in the data. When attaching constraints to some class, the natural way to invoke validation is to examine each typed object and test it against the constraints corresponding to those types. If your constraints don't happen to be the product of all of the constituent classes, you need to invent a discriminating type, a new type assertion that connects the instance data to constraints that apply to its use in some context. This approach either requires the instance data to include discriminating types for all of its potential consumers or some other way to infer the new types from the existing structure. Of course, if those constraints don't apply in all contexts, you have a complex system where one must assert the discriminating type for some process, validate for that process and then strike either the type assertion or the constraints from view before examining that same data for use in a different context. Once again, the tension appears to lie fundamentally in the mode of use of RDF, optimizing for reuse vs. centralized controls and requirements.

Path Forward

An ideal shapes model would be intuitive and familiar to SPIN users at the same time as meeting the goals of reusable types and nuanced validation. The concept of record classes appears to be a promising basis for consensus within the RDF Data Shapes Working Group. The contextual validation that drove the development of Resource Shapes, Shape Expressions and Description Set Profiles are all easily satisfied by having a shape object and a variety of ways to apply a shape to some objects. Some of these ways may be expressed in RDF, but, just as with XML validation, it's not necessary that all triggers have a defined representation in the data. If someone wants to use the same identifier for shapes and classes, they can easily signal that with a classShape property pointing back at the same node, but in the future they will have more controls over the association between data and the shapes that constrain that

data. This solution provides everyone with a useful new tool – shape validation – which they can apply in as nuanced a manner as they see fit.

It is easy to criticize requirements on discriminating types as arcane and unnecessary, but it reflects a real deployment. Most early iterations in extreme programming appear naive after one extends the system to deal with more use cases, just as more developed models appear complex to those with the use cases met by the earlier model. The RDF Data Shapes WG can profit from the experience of SPIN and ICV. New iterations of SPIN can provide users with more nuanced ways to initiate validation that avoids the creation of mutually exclusive constraints.

Acknowledgement: The authors gratefully acknowledge extensive and helpful textual suggestions and conceptual clarifications to a draft of this article from Peter Patel-Schneider. ■