Requirements on RDF Constraint Formulation and Validation

Thomas Bosch[1] and Kai Eckert[2]

[1] GESIS – Leibniz Institute for the Social Sciences, Germany
thomas.bosch@gesis.org,
[2] University of Mannheim, Germany
kai@informatik.uni-mannheim.de

Abstract. For many RDF applications, the formulation of constraints and the automatic validation of data according to these constraints is a much sought-after feature. In 2013, the W3C invited experts from industry, government and academia to the RDF Validation Workshop, where first use cases have been presented and discussed. In collaboration with the W3C, a working group on RDF Application Profiles (RDF-AP) is currently established in the Dublin Core Metadata Initiative that follows up on this workshop and addresses among others RDF constraint formulation and validation.
In this paper, we present a database of requirements obtained from various sources, including the use cases presented at the workshop as well as in the RDF-AP WG. The database, which is openly available and extendible, is used to evaluate and compare several existing approaches for constraint formulation and validation. We present a classification and analysis of the requirments, show that none of the approaches satisfy all requirements and aim at laying the ground for future work, as well as fostering discussions how to close existing gaps.

Keywords: RDF Constraint Formulation, RDF Constraint Validation, Requirements, OWL2, RDF, Linked Data, Semantic Web

1 Introduction

The notion of Linked (Open) Data and its principles clearly increased the acceptance – not to say the excitement – of data providers for the underlying Semantic Web technology. Early concerns of the data providers regarding stability and trustability of the data have been addressed and largely been solved, not only by technical means regarding versioning and provenance, but also by the providers getting accustomed to the open data world with its pecularities.

Linked Data and RDF, however, still are not the primary means to create, store, and manage data on the side of the providers. Linked Data is mostly provided as a view on data, a one-way road, disconnected from the internal data representation. To the obstacles for full adoption of RDF, possibly comparable to XML, belong the lack of accepted ways to formulate (local) constraints on

data and to validate data. The W3C reports a consensus among 27 participants from industry, government and academia of RDF Validation Workshop[3] that there are the following needs:

1. Declarative definition of the structure of a graph for validation and description.
2. Extensible to address specialized use cases.
3. A mechanism to associate descriptions with data.

Several use-cases with requirements have been presented at the workshop, further requirements are described in talks about general approaches and experiences outside of RDF, like Dublin Core Application Profiles or XML Schema Definitions. An important finding is that there are non-functional requirements for data validation in a Linked Data setting, particularly the need to "communicate the constraints against which data is to be validated in a way which is both easy to understand by human beings and discoverable by programs."

SPARQL and SPIN are powerful and widely used for constraint formulation and validation [2], but constraints formulated as SPARQL queries are not as understandable as one wishes them to be. Consider the following example of the simple constraint stating that only dogs are allowed as pets:

```
SELECT ?this ?subope ?object WHERE {
    ?C owl:allValuesFrom :Dog .
    ?C owl:onProperty :hasPet .
    ?C a owl:Restriction .
    ?this rdf:type ?subC . ?subC rdfs:subClassOf* ?C .
    ?this ?subOPE ?object . ?subOPE rdfs:subPropertyOf* :hasPet .
    FILTER NOT EXISTS { ?object rdf:type :Dog . } }
```

This query checks the constraint and returns violating triples, but the actual constraint could be formulated much shorter, for instance using the OWL 2 Functional-Style syntax:

```
SubClassOf( :strictDogOwner ObjectAllValuesFrom( :hasPet :Dog ) )
```

Similarly, but even shorter, as Shape Expression:

```
<StrictDogOwnerShape> { :hasPet :Dog+ }
```

Partly as follow-up to the W3C workshop and partly due to further expressed requirements at the Semantic Web in Libraries conference 2013[4], the Dublin Core Metadata Initiative in collaboration with the W3C currently establishes a Working Group for RDF Application Profiles (RDF-AP WG) that will investigate existing approaches and best-practices, identify possible gaps and propose

[3] RDF Validation Workshop – Practical Assurances for Quality RDF Data. 10-11 September 2013, Cambridge, MA, USA. http://www.w3.org/2012/12/rdf-val/report

[4] SWIB13 – Semantic Web in Libraries, 25 - 27 November 2013, Hamburg, Germany. http://swib.org/swib13/

practical solutions for the representation of application profiles, including the formulation of data constraints.[5] The RDF-AP WG bases its work on currently 8 case studies and use cases provided by internal and external stakeholders, mostly from the library domain. In a heterogenous environment like the Web, there is not necessarily a one-size-fits-all solution, especially as existing solutions should rather be integrated than replaced, not least to avoid long and fruitless discussions about the "best" approach.

Our work presented in this paper is supposed to lay the ground for subsequent activites in the working group. Our contributions are two-fold: first, we propose to relate existing solutions to specific case-studies and use-cases by means of requirements extracted from the latter and fulfilled by the former. We therefore created and present an exhaustive database of all requirements identified in the validation workshop and the RDF-AP WG. Additionally, we added requirements from other sources, particularly in the form of constraint types that are supported by existing approaches, e.g., expressable in OWL2.

Second, we use this database to provide an overview on different classes of requirements and give examples, to what degree these classes of requirements are supported by different approaches. We want to highlight strengths and weaknesses of these approaches and identify gaps and possible solutions for their elimination.

2 From a Case Study to a Solution (and Back)

In the development of standards, as in software, case studies and/or use cases are usually taken as starting point. In case studies, the full background of a specific scenario is described, where the standard or the software is to be applied. Use cases are smaller units where a certain action or a typical user enquiry is described. They can be extracted from and thus linked to case studies, but often they are defined directly.

Requirements are extracted from use cases; they form the basis for development and are used to test the result. We specifically use the requirements to evaluate existing approaches for constraint formulation and validation. Via the requirements, the approaches get linked to use cases and case studies and it becomes visible which approaches can be used in a given scenario and what drawbacks might be faced.

We classify the requirements to provide a high-level view on different approaches and to facilitate a better understanding of the problem domain. Our database is openly available and can be extended with new case studies, use cases, requirements and approaches.

Table 1 shows an excerpt from our database. The general structure is a polyhierarchy from case-studies over use-cases and requirements to solutions. All instances contain at least uplinks to the next level, i.e., solutions are linked to requirements that they fulfill and possibly requirements that they explicitly

[5] http://wiki.dublincore.org/index.php/RDF-Application-Profiles

Table 1. Database Examples

ID	Title	Links	Description
Case Studies			
CS-1	DPLA	UC-1	The Digital Public Library of America maintains an access portal to digitized cultural heritage objects... We harvest data using several different methods...[6]
Use Cases			
UC-1	Recommend Property	CS-1	Some properties may not be mandatory, but may be recommended to indicate a "value-added" level of compliance with MAPv3...
Requirements			
R-1	Optional Properties	UC-1	A property can be marked as optional. Valid data MAY contain the property.
R-2	Recommended Properties	UC-1, R-3	An optional property can be marked as recommended. A report of missing recommended properties is generated. Fulfilled if R-3 is fulfilled.
R-3	Classified Properties	UC-1	A custom class like "recommended" or "deprecated" can be assigned to properties and used for reporting.
Solutions			
S-1	ShEx	R-1/2/3	Fulfilled: R-1 (minimum cardinality = 0, maximum cardinality = 1). Not fulfilled: R-2, R-3.
S-2	SPIN	R-1/2/3	Fullfilled: R-1, R-2, R-3.

do not fullfill. Requirements are linked to use-cases, which are linked to case studies.

The polyhierarchy allows the linking of all elements to more than one parent, requirements particularly are linked to several use cases. Our goal is to maintain a set of distinct requirements. Only this way it is possible to evaluate the solutions regarding their suitability for the use cases and case studies in our database. Use cases can be shared between case studies as well, but this is harder to maintain as use cases are less formal and often more case specific than a requirement.

Requirement R-2 is an example, where a link between requirements is established. In this case, the link is used to point to a requirement that is "broader" than this requirement, i.e., should that requirement be fulfilled, then this requirement is automatically fulfilled as well. In a similar way requirements can be linked to duplicates if they should occur. Our goal is a relative stability regarding the requirements, which then can prove useful to mediate between data and solution providers.

The database is made available at http://purl.org/net/rdf-validation. The initial database was created manually and forms the basis of this paper. The web application to access the database is currently in a beta state and still under development. Nevertheless, the full database can already be browsed online and interested participants can register and contribute to the database.

3 Related Work

Requirements Engineering is recognized as a crucial part of project and software development processes. Similar to our collaborative effort, Lohmann et al. propose social requirements engineering, i.e. the use of social software like wikis to support collaborative requirements enginieering [4]. Their approach focuses on simplicity and supports in particular the early phases of requirements engineering with many distributed participants and mainly informal collaboration. They emphasize the social experience of developing requirements for software systems: Stakeholders are enabled to collaboratively collect, discuss, improve, and structure requirements. Under the supervision of experts, the requirements are formulated in natural language and are improved by all participants step by step. Later on, experienced engineers may clean and refine requirements. As basis for their work, they developed a generic approach (Softwiki) using semantic technologies and the SWORE ontology for capturing requirements relevant information semantically [5]. The SWORE ontology, as well as a prototypical implementation of their approach is available online.[7] We evaluated the implementation and the ontology regarding a possible reuse, but it turned out that Softwiki focuses clearly on the requirements within a traditional software development process, while we need a broader view including case studies, use cases and various implementing approaches. Nevertheless we will reuse parts of the SWORE ontology and include links wherever possible.

To the best of our knowledge, their is no comparable prior work regarding the collection of a comprehensive list of requirements for the formulation and validation of constraints, neither exist general approaches to *compare* different solutions based on common or differing requirements. More related work focuses on specific constraint languages and implementations, which we will introduce in the next section.

4 Approaches for Constraint Formulation and Validation

In this section, we present current approaches for constraint formulation and validation which have been the most discussed in the mentioned workshops and WGs. These approaches differ in 2 dimensions: (1) the used constraint language and (2) if they offer validation systems.

OWL, Resource Shapes (ReSh), Shape Expressions (ShEx), Description Set Profiles (DSPs), SPARQL, and SPIN are the most promising and applied constraint languages. Stardog ICV, Pellet ICV, and SPIN use OWL 2 constructs to formulate constraints. SPIN[8] provides a vocabulary to represent SPARQL queries as RDF triples and uses SPARQL to specify inference rules and logical constraints [2]. The Pellet Integrity Constraint Validator (ICV)[9] is a proof-of-concept extension for the OWL reasoner Pellet. Stardog ICV[10] validates RDF

[7] http://softwiki.de/netzwerk/en/
[8] http://spinrdf.org
[9] http://clarkparsia.com/pellet/icv/
[10] http://docs.stardog.com/icv/icv-specification.html

data stored in a Stardog RDF database. ReSh[11] defines its own RDF vocabulary Open Services for Lifecycle Collaboration (OSLC) to define constraints [6]. ShEx[12] also specifies a new constraint language whose syntax and semantics are similar to regular expressions. DCMI RDF Application Profile (AP)[13] and Bibframe[14] are approaches to specify profiles for application-specific purposes. DCMI RDF-AP uses DSP[15] as generic constraint language which is also intuitive for non-experts. The Bibframe constraint language has a strong overlap with DSP. Kontokostas et al. define 17 data quality integrity constraints represented as SPARQL query templates called Data Quality Test Patterns (DQTP) [3]. Schemarama[16] is based on the Squish RDF language instead of SPARQL. For XML, Schematron[17] is an ISO standard for validation and quality control of XML documents based on XPath and XSLT. XML Schema[18] is the primary technology for specifying and constraining the structure of XML documents.

In addition to constraint validation languages, SPIN (open source API), Stardog ICV (as part of the Stardog RDF database), DQTP (tests), Pellet ICV (extension of Pellet OWL reasoner) and ShEx offer executable validation systems using SPARQL as implementation language.

In this paper, we evaluate to which extend these approaches cover classes of requirements (1) to express different types of constraints and (2) to formulate constraints. For the formulation of constraints, it is important that the constraint language is concise and intuitive and that the declarative constraint language is translated to an implementation language like SPARQL in order to execute constraint validation automatically. In form of concrete examples, we show how current approaches can be used to express different types of constraints and how they can be used together to fulfill the majority of the identified requirements classes.

5 Requirements

Use cases discussed within the scope of the mentioned workshops and working groups leaded to the definition of requirements on RDF constraint formulation and validation. We classified these requirements into the 2 top-level categories 'Constraint Formulation' and 'Constraint Expressivity'.

5.1 Formulation of Constraints

Intuitive and concise language. We claim that all constraints can be expressed using the low-level language SPARQL. The majority of the constraints

[11] http://www.w3.org/Submission/shapes/
[12] http://www.w3.org/2013/ShEx/Definition
[13] http://dublincore.org/documents/singapore-framework/
[14] http://bibframe.org/
[15] http://dublincore.org/documents/dc-dsp/
[16] http://swordfish.rdfweb.org/discovery/2001/01/schemarama/
[17] http://www.schematron.com/
[18] http://www.w3.org/TR/xmlschema-1/

can also be written more declaratively, intuitively, and concisely in form of OWL 2 axioms in the concrete syntax Turtle. Although, OWL 2 is a very expressive language, we cannot express every constraint in OWL 2. The succeeding existential quantification contains those individuals that are connected by the `:fatherOf` property to individuals that are instances of the class `:Man`. The ontology, the constraint, and RDF data are expressed with the same OWL 2 axiom and the same concrete syntax:

```
[ rdfs:subClassOf [
  a owl:Restriction;
  owl:onProperty :fatherOf;
  owl:someValuesFrom :Man ] ] .
```

The main purpose of OWL 2 is to infer new knowledge from existing schemata and data rather than to check data for inconsistencies. Therefore, most constraint validation approaches define constraints with other high-level declarative languages, even though most people are familiar with OWL 2 and its concise human-understandable concrete syntax Turtle. OWL 2 can be used to describe RDF data, to infer new knowledge, and to validate RDF data using the same expressive OWL 2 axioms. With XML Schemas, we also structure and validate our data according to that structure.

Shape Expressions contain elements from regular expressions making the language concise and intuitive. In the following example, an employee has at least 1 given name, 1 family name, any number of phone numbers, and 1 mail box:

```
<EmployeeShape> {
    foaf:givenName xsd:string+ ,
    foaf:familyName xsd:string ,
    foaf:phone IRI* ,
    foaf:mbox IRI }
```

As different constraints can be expressed with different languages, we propose to use multiple languages to define constraints depending on the requirements which have to be satisfied.

Translated to implementation language. High-level declarative languages like OWL 2 cannot be executed directly to validate constraints. Therefore, we take a low-level execution language like SPARQL. Sirin and Tao showed how constraints can be translated to nonrecursive Datalog programs for validation [7] and Angles and Gutierrez explained that SPARQL has the same expressive power as nonrecursive Datalog programs [1]. As a consequence, we can also use SPARQL queries to validate constraints. Thus, constraint validation can be reduced to SPARQL query answering. The participants of the 2013 W3C RDF Validation workshop agreed that SPARQL should be the language to execute constraint validation[19]. Furthermore, all evaluated constraint validation approaches execute constraint validation with SPARQL. The next SPARQL query shows how the OWL 2 existential quantification is implemented in SPIN:

[19] http://www.w3.org/2013/09/10-rdfval-minutes

```
CONSTRUCT {
    _:violation
        a spin:ConstraintViolation ;
        rdfs:label ?violationMessage
        spin:violationRoot ?this }
WHERE {
    ?this rdf:type ?subC . ?subC rdfs:subClassOf* ?C .
    ?C owl:someValuesFrom ?CE .
    ?C owl:onProperty ?OPE .
    ?C a owl:Restriction .
    FILTER ( sp:not ( spl:hasValueOfType ( ?this, ?OPE, ?CE ) ) ).
    FILTER EXISTS { ?this ?OPE ?object . ?object rdf:type owl:Thing . }
    BIND ( ( ... ) AS ?violationMessage ) . }
```

RDF representation of constraints. One of the main benefits of SPIN is that arbitrary SPARQL queries and thus constraints are represented as RDF triples. SPIN provides a vocabulary, the SPIN SPARQL Syntax, to represent SPARQL queries in RDF. The benefits of an RDF representation of constraints are:

- constraints can be consistently stored together with ontologies and RDF data
- constraints can be easily shared on the web of data
- constraint validation can be executed automatically
- constraints can be processed by a plethora of already existing RDF tools
- constraints are linked to RDF data

The subsequent code snippet demonstrates how SPIN represents SPARQL 1.1 NOT EXISTS filter expressions in RDF:

```
FILTER NOT EXISTS { ?person foaf:name ?name }
-----
[   a sp:Filter ;
    sp:expression [
        a sp:notExists ;
        sp:elements (
            [   sp:subject [ sp:varName "person" ] ;
                sp:predicate foaf:name ;
                sp:object [ sp:varName "name" ] ] ) ] ] )
```

Our approach, which is implemented in Java, executes constraint validation with SPIN. SPIN templates define the validation of both OWL 2 constraints and constraints only expressible with SPARQL. These constraints are checked for each resource of the type owl:Thing (all resources are assigned to the common super-class owl:Thing).

Constraint validation results. Like ontologies, instance data, and constraints, we should also represent constraint violations in RDF. SPIN templates construct (SPARQL CONSTRUCT) constraint violation triples containing information about constraint violations, which cannot be expressed directly in OWL 2:

```
CONSTRUCT {
    _:icViolation
        a spin:ConstraintViolation ;
```

```
        rdfs:label ?violationMessage ;
        spin:violationRoot ?violationRoot ;
        spin:violationPath ?violationPath ;
        spin:violationSource ?violationSource ;
        spin:fix ?violationFix ;
        :severityLevel ?severityLevel }
```

Constraint violations (of the type `spin:constraintViolation`) should provide a useful message (`rdfs:label`) explaining the reasons why the data did not satisfy the constraints, which aids data debugging and repair. If we do not state the triples `:Peter :fatherOf :Stewie .` and `:Stewie a :Man .`, the SPIN template checking the OWL 2 existential quantification on the object property `:fatherOf` constructs a constraint violation triple raising the message 'ObjectSomeValuesFrom(:fatherOf :Man) - :Stewie must be an instance of :Man'. Now, you know exactly why the data violated this constraint and you know where you have to modify your data. Constraint violation triples contain references to triples causing the constraint violations (`spin:violationRoot`) and references to constraints causing constraint violations (`spin:violationSource`). In our example, the subject `:Peter` causes the constraint violation and the constraint `:ObjectSomeValuesFrom` constructs the constraint violation triple. To fix constraint violations we need to give some guidance how to become valid data (`spin:fix`). Appropriate triples may point to useful messages explaining in detail how to overcome constraint violations. Constraint violations can be classified according to different levels of severity (`:severityLevel` having controlled vocabulary as range with elements like `:Error` and `:Warning`). It is also important to find not validated triples, i.e. triples which have not been validated by any constraint, as it may be enforced that every triple of the data have to be validated.

5.2 Constraint Expressivity

Cardinality Restrictions. Class expressions in OWL 2 can be formed by placing restrictions on the cardinality of object and data property expressions. All cardinality restrictions can be qualified or unqualified. The class expressions contain those individuals that are connected by a property expression to at least, at most, and exactly a given number of instances of a specified class expression. Qualified and unqualified cardinality restrictions can be expressed in OWL 2:

```
:CE rdfs:subClassOf [
    a owl:Restriction ;
    owl:maxQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
    owl:onProperty :hasSon ;
    owl:onClass :Man ] .
:Peter a :CE ;
    :hasSon :Stewie [ a :Man ] .
```

`:Peter` is an instance of the class expressions containing those individuals having at most 1 son which is `:Stewie` in the RDF instance data. If we state that `:Peter` has a second son or if we do not assign `:Stewie` to the class `:Man`, the qualified maximum cardinality restriction will be violated. SPIN, Stardog, and

Shape Expressions are the only approaches with witch qualified and unqualified cardinality restrictions on data and object properties can be specified.

Disjointness. Disjointness of classes and union of class expressions, (class-specific) object and data properties, and individuals is a very important type of constraints which can be completely covered with SPIN (implementing OWL 2 constructs). An OWL 2 disjoint union axiom DisjointUnion(C CE_1 ... CE_n) states that a class C is a disjoint union of the class expressions CE_i, $1 \leq i \leq n$, all of which are pairwise disjoint. Each instance of C is an instance of exactly one CE_i, and each instance of CE_i is an instance of C[20]. According to the next disjoint union of 2 class expressions, each child is either a boy or a girl, each boy is a child, each girl is a child, and nothing can be both a boy and a girl. As in this example, `:Stewie` is both a boy and a girl, a constraint violation is raised:

```
:Child owl:disjointUnionOf ( :Boy :Girl ) .
:Stewie a :Child ; a :Boy ; a :Girl .
```

Disjoint groups of object and data properties can be expressed in OWL 2:

```
[   rdf:type owl:Class ;
    owl:unionOf (
        [   rdf:type owl:Restriction ;
            owl:qualifiedCardinality 1  ;
            owl:onProperty foaf:name ;
            owl:onClass xsd:string ]
        [   rdf:type owl:Class ;
            owl:intersectionOf (
                [   rdf:type owl:Restriction ;
                    owl:minQualifiedCardinality 1 ;
                    owl:onProperty foaf:givenName ;
                    owl:onClass xsd:string ] .
                [   rdf:type owl:Restriction ;
                    owl:qualifiedCardinality 1 ;
                    owl:onProperty foaf:familyName ;
                    owl:onClass xsd:string ] ) ] ) ] .
```

In this example, we define a shape for persons. A person has either a FOAF name or 1 or more given names and 1 family name. Although this kind of constraint can be realized in OWL 2, the definition of disjoint groups of properties is not that intuitive and declarative. Exactly the same constraint can be expressed more concisely with Shape Expressions:

```
<PersonShape> {
  ( foaf:name xsd:string
    |
    foaf:givenName xsd:string+ ,
    foaf:familyName xsd:string ) }
```

Shape Expressions and SPIN are the only approaches to specify disjoint groups of properties for given classes.

Constraints on RDF Properties. Object as well as data properties may be constrained. The main component of an OWL 2 ontology is a set of axioms

[20] http://www.w3.org/TR/owl2-syntax/

- statements that say what is true in the domain. OWL 2 provides axioms that can be used to characterize and establish relationships between object and data property expressions. An object property functionality axiom states that an object property expression is functional - that is, for each individual x, there can be at most one distinct individual y such that x is connected by the object property expression to y[21]. With Pellet ICV, we can state a couple of object and data property axioms like the following object property functionality axiom in OWL Turtle syntax [7]:

```
:isManufacturedBy a owl:FunctionalProperty .
:Product :isManufacturedBy :Manufacturer1 , :Manufacturer2 .
```

The object property `:isManufacturedBy` is defined as functional. The OWL interpretation would infer that the manufacturers are the same resources, as nothing contradicts the inference that these two manufacturers are the same and there is no Unique Name Assumption. With constraint semantics, however, a constraint violation is raised. With Resource Shapes 2.0 and Shape Expressions it is not possible to declare functionality axioms on object and data properties. We can define these axioms with SPIN (and OWL 2), Stardog, and Pellet.

Object property paths (supported by Stardog and SPIN) are important constraints within various domains. Object property chains can be expressed as OWL 2 axioms SubObjectPropertyOf(ObjectPropertyChain(OPE_1 ... OPE_n) OPE) stating that, if an individual x is connected by a sequence of object property expressions OPE_1, ..., OPE_n with an individual y, then x is also connected with y by the object property expression OPE[22]. As the triple `:Stewie :hasAunt :Carol .` is not contained in the following data set, a constraint violation results:

```
:hasAunt owl:propertyChainAxiom ( :hasMother :hasSister ) .
:Stewie :hasMother :Lois . :Lois :hasSister :Carol .
```

Constraints on RDF objects. For RDF objects, we can state constraints such as allowed values, default values, and negative object constraints. Resource Shapes 2.0 enables defining allowed values for RDF objects as well as RDF literals:

```
:oslc-change-request a oslc:ResourceShape ;
    oslc:property :oslc_cm-status .
:oslc_cm-status a oslc:Property ;
    oslc:allowedValues :status-allowed-values .
:status-allowed-values a oslc:AllowedValues ;
    oslc:allowedValue "Done" , "InProgress" , "Submitted" .
```

The constraint above specifies the only allowed values of the status data property for change request resources. If change requests have other status values, constraint violations will be raised. In addition to Resource Shapes 2.0, the DCMI

[21] http://www.w3.org/TR/owl2-syntax
[22] http://www.w3.org/TR/owl2-syntax

RDF-APs and SPIN (and OWL 2) allow specifying allowed values for RDF literals. For RDF objects, we can apply the approaches Resource Shapes 2.0, Shape Expressions, DCMI RDF-APs, and SPIN (and OWL 2) to define allowed values.

With DCMI RDF-APs and SPIN, we can declare that RDF objects and literals have to be part of specific controlled vocabularies. These statements are represented with DCMI RDF-APs using an RDF triple comprising an RDF subject that is the value RDF node, an RDF predicate `dcam:memberOf`, and an RDF object with a corresponding RDF URI Reference being the DCAM vocabulary encoding scheme URI[23]. The following excerpt states that a given book is assigned to the topic 'Ornitology' which is part of a particular controlled vocabulary:

```
:Book
    dcterms:subject [
        rdf:value "Ornitology" ;
        dcam:memberOf :ControlledVocabulary ] .
```

Constraints on RDF Literals. Constraint on RDF literals are not that significant in the Linked Data community, but they are very important in communities like the library domain. For RDF literals, range-specific, constraining facet-specific, datatype-specific constraints, and language-specific can be defined. We can restrict the datatypes, RDF literals have to correspond to, with XML Schema constraining facets. SPIN allows us to implement all constraining facets. DQTPs enables constraining literal values to match or not to match a certain regex pattern (`xsd:pattern`):

```
SELECT DISTINCT ?s WHERE { ?s %% P1 %% ? value .
    FILTER ( %% NOP %% regex (str (? value ), %% REGEX %) ) }
```

`P1` is the property we need to check against `REGEX` and `NOP` can be a not operator (!) or empty. An example binding could be to check if the `dbo:isbn` format is different (!) from "^([iIsSbBnN 0-9-])*$" [3]. DQTPs also enables constraining literal values (having a certain datatype) to be or not to be within a specific range (`xsd:maxInclusive, xsd:maxExclusive, xsd:minExclusive, xsd:minInclusive`):

```
SELECT DISTINCT ?s WHERE {
    ?s rdf:type %% T1 %% . ?s %% P1 %% ?value .
    FILTER ( %% NOP %% (?value < %% Vmin %% || ?value > %% Vmax %%))) }
```

For instance, we can restrict geographical longitudes and latitudes (`geo:lat`, `geo:long`) of a spatial feature to be within the range [-90,90][3]. Furthermore, we implemented the constraining facet `xsd:whiteSpace` in SPIN to avoid leading and trailing white spaces in literals. Sub-types of language-specific constraints on RDF literals are constraints (1) to check if a literal for a specific data property within the context of a particular class has a given language tag, (2) to check whether the literal, within the context of a given property and class, is missing,

[23] http://dublincore.org/documents/dc-rdf/

or (3) to ensure that resources of a given type must have at most 1 value of a specific language for a given data property (e.g. a single English ("en") rdfs:label). Default values can be defined with Bibframe, Resource Shapes 2.0, and SPIN. For this purpose, SPIN constructors may contain SPARQL CONSTRUCT queries for specific classes (e.g. USA is the birth country of each `USCitizen`):

```
:USCitizen a rdfs:Class ;
    spin:constructor [ a sp:Construct ; sp:text """
    CONSTRUCT { ?this :birthCountry "USA" . } WHERE {} """ ] .
```

6 Evaluation

In this section, we evaluate current approaches according to the top-level classification of constraint validation requirements. This kind of evaluation is crucial for future improvements regarding constraint formulation and validation of both existing and new approaches. The underlying facts result primarily from the individual official specifications. We categorize requirements classes to see which requirements are well, badly, and limited satisfied by which approaches. The goal of this evaluation is not to completely evaluate all currently available constraint validation approaches. We want to show in a generic way that none of the current approaches satisfies all requirements and that different approaches cover different requirements classes. Case studies and use cases define what requirements classes have to be covered. This evaluation indicates which approaches to use to cover specific requirements classes and therefore use cases. There are 2 first level requirements classes: 'constraint expressivity' and 'constraint formulation'. Table 2 and 3 show for each approach what second level requirements classes are covered to which extend. Numbers in brackets behind requirements classes indicate the amount of requirements contained in that class. Numbers in brackets in table cells indicate that requirements are limited satisfied.

Good Coverage. Although equivalence (e.g. equivalent classes) is only considered by 1 approach (SPIN), all 4 associated requirements are satisfied. 1 approach (SPIN) covers all 20 requirements on RDF properties constraints (e.g. object property paths) and 2 approaches (DQTP and Stardog) fulfill half of these requirements. Just 1 approach (SPIN) covers 4 of 5 identification requirements (e.g. to check if IRIs correspond to specific patterns). Class expressions represent sets of individuals by formally specifying conditions on the individuals' properties; individuals satisfying these conditions are said to be instances of the respective class expressions. Sub-categories of this requirements class are well satisfied by 3 approaches (DQTP, Shape Expressions, and Stardog) and nearly exhaustively satisfied by 1 approach (SPIN). Class-relationships (e.g. subsumption) and set-oriented operations (e.g. negation of classes) are not supported by many approaches. In contrast, property occurrences (e.g. mandatory or optional), property restrictions (e.g. existential quantifications), and cardinality restrictions are supported by the majority of current approaches. Constraints on individuals (e.g. negative object property assertions) are only considered by 1 approach (SPIN) which fulfills all associated requirements.

Table 2. Constraint Expressivity

Requirement Classes	BF	DCMI	DQTP	Pellet	RS	SE	SPIN	Stardog
Disjointness (8)	✗	✗	3		✗	3	5	
Equivalence (4)	✗	✗			✗	✗	4	
Constraints on RDF properties (20)			12	3	1(1)	2	20	7
Constraints on RDF objects (7)	2	2	1	1	3(1)	5	5	2
Constraints on RDF literals (14)	2	2	4		3(1)	2(1)	7	
Identification (5)	✗	✗			✗	(2)	4	✗
Uniqueness (2)							1	
Provenance Constraints	✗	✗	✗	✗	✗	✗	✗	✗
Constraints on Individuals (6)	✗	✗			✗	✗	6	
Class Relationships (4)			2			1	4	1
Set-Oriented Operations (6)				2			6	3
Property Occurrences (9)	1	1	1		3	6	6	2
Property Restrictions (10)			1	2		2	8	3
Cardinality Restrictions (12)	✗	✗	6	✗	(12)	12	12	3

Limited Coverage. Approach developers should mention requirements which are not covered exhaustively by current approaches. Only 3 approaches (DQTP, Shape Expressions, and SPIN) consider disjointness constraints (e.g. class-specific disjoint property groups) and 1 approach (SPIN) covers 5 of 8 disjointness requirements. 5 of 7 requirements on RDF objects constraints (e.g. allowed values) can be expressed with 2 approaches (Shape Expressions and SPIN). There are 2 requirements to ensure uniqueness (e.g. unique URIs), but only 1 approach (SPIN) satisfies 1 requirement. Other approaches do not cover uniqueness requirements.

Bad Coverage. For future development of approaches it is crucial to especially consider requirements which are currently not satisfied at all by any approach. So far, provenance constraints are not considered by approach developers. Most approaches satisfy just 2 of 14 requirements on RDF literal constraints (e.g. range of literal values). At least 1 approach (SPIN) covers 50% of these requirements.

Table 3 shows constraint formulation requirements (classes) and their coverage by current approaches. Even though, almost each constraint language is intuitive, only 4 constraint languages can be seen as both intuitive and concise (Pellet, Shape Expressions, SPIN, and Stardog). 3 of these 4 approaches use OWL 2 as declarative language - the standard language to define ontologies. Shape Expressions uses a language similar to regular expressions.

5 of 8 approaches translate declarative constraints formulations to an implementation language (e.g. SPARQL) to execute constraint validation. It is very important for future enhancements by the whole community that implementations are not only existent but also publicly available. 5 of 8 approaches are implemented, but implementations are publicly available for only 2 approaches (public availability of implementations is limited for 2 further approaches). Constraints are represented as RDF triples by only 1 approach (SPIN). RDF should

Table 3. Constraint Formulation

Requirement Classes	BF	DCMI	DQTP	Pellet	RS	SE	SPIN	Stardog
Intuitive Language	✓	✓	∼	✓	✓	✓	✓	✓
Concise Language	✗	✗	✓	✓	✗	✓	✓	✓
Translated to Implementation Language	✗	✗	✓	✓	✗	✓	✓	✓
Implemented Constraint Validation	✗	✗	✓	✓	✗	✓	✓	✓
Implementation Publicly Available	✗	✗	✓	∼	✗	✗	✓	∼
RDF Representation of Constraints	✗	✗	✗	✗	✗	✗	✓	✗
Constraint Validation Results (10)	✗	✗	2	2	✗	6	9	1

be the natural and standard format to represent constraints within the Linked Data community. 2 approaches (Shape Expressions and SPIN) cover almost all requirements on validation results (e.g. providing some guidance how to become valid data). Unfortunately, 3 of the remaining approaches cover requirements on validation results very poorly.

7 Conclusions and Future Work

Heterogenous approaches with different strengths and weaknesses are not a bad thing; we do not expect there to be a one-size-fits-all solution, nor do we aim at creating one. With this paper, we rather want to raise the awareness towards the differences and commonalities of existing approaches as well as to shed some light on the different requirements that data providers currently have. Therefore, we presented our approach to collect case studies, use cases and especially requirements collaboratively and in structured form. By linking the requirements to existing constraint languages and validation systems, we could identify strengths and weaknesses, commonalities and differences not only intellectually, but based on reliable data.

The main purpose of this work is to support discussions of the different approaches and to help stakeholders in the choice or in the development of appropriate solutions. In the context of application profiles, where the publication of constraints together with the data model is crucial, we want to emphasize the need for concise, easy to understand constraint languages. This requirement is often neglegtected in discussions of approaches. While consistency is understandably desired, it has to be questioned if one constraint language can fulfill all requirements without being overly complicated or if different approaches should rather be used for different classes of requirements. This holds especially for different levels of abstraction, as the possibility to define constraints on the format of RDF literals compared to constraints on the availability or special properties of provenance information. Both represent examples where all current approaches lack proper support.

Gaps within a class of requirements, e.g., disjointness, constraints on RDF objects, or uniqueness, should be easier to close within the existing approaches. This would lead to a harmonization of the approaches ragarding their expressivity and enable translations in-between or towards a general constraint language,

e.g., the translation of well-readable constraints in any language to executable SPARQL queries. The latter is especially promising considering that SPARQL is able to fulfil all functional requirements and already considered by many as a practical solution to formulate constraints.

As future work, we plan to provide a complete implementation of OWL 2 constraints in form of SPIN templates to demonstrate this approach. We will extend and maintain the requirements database and hope to establish it as an important tool for the advancement of constraint formulation and validation in RDF. Within the DCMI RDF Application Profiles Working Group, we pursue the establishment of application profiles that among others allow to link constraints directly to published datasets and ontologies.

References

1. Renzo Angles and Claudio Gutierrez. The expressive power of SPARQL. In *Proceedings of the 7th International Semantic Web Conference (ISWC2008)*, pages 114–129, 2008.
2. Christian Fürber and Martin Hepp. Using SPARQL and SPIN for Data Quality Management on the Semantic Web. In Witold Abramowicz and Robert Tolksdorf, editors, *Business Information Systems*, volume 47 of *Lecture Notes in Business Information Processing*, pages 35–46. Springer Berlin Heidelberg, 2010.
3. Dimitris Kontokostas, Patrick Westphal, Sören Auer, Sebastian Hellmann, Jens Lehmann, Roland Cornelissen, and Amrapali Zaveri. Test-driven evaluation of linked data quality. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 747–758, Republic and Canton of Geneva, Switzerland, 2014. International World Wide Web Conferences Steering Committee.
4. Steffen Lohmann, Sebastian Dietzold, Philipp Heim, and Norman Heino. A web platform for social requirements engineering. In Jürgen Münch and Peter Liggesmeyer, editors, *Software Engineering (Workshops)*, volume 150 of *LNI*, pages 309–315. GI, 2009.
5. Steffen Lohmann, Philipp Heim, Sören Auer, Sebastian Dietzold, and Thomas Riechert. Semantifying requirements engineering – the softwiki approach. In *Proceedings of the 4th International Conference on Semantic Technologies (I-SEMANTICS '08)*, J.UCS, pages 182–185, 2008.
6. Arthur G. Ryman, Arnaud Le Hors, and Steve Speicher. Oslc resource shape: A language for defining constraints on linked data. In Christian Bizer, Tom Heath, Tim Berners-Lee, Michael Hausenblas, and Sören Auer, editors, *LDOW*, volume 996 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.
7. E. Sirin and J. Tao. Towards integrity constraints. In *Proceedings of the Workshop on OWL: Experiences and Directions, OWLED 2009*, 2009.