DCPAPERS
Proc. Int'l Conf. on Dublin Core and Metadata Applications 2015

Guidance, Please! Towards a Framework for RDF-based Constraint Languages

Thomas Bosch
GESIS – Leibniz Institute
for the Social Sciences, Germany
thomas.bosch@gesis.org

Kai Eckert
Stuttgart Media University, Germany
eckert@hdm-stuttgart.de

Abstract

In the context of the DCMI RDF Application Profile task group and the W3C Data Shapes Working Group solutions for the proper formulation of constraints and validation of RDF data on these constraints are being developed. Several approaches and constraint languages exist but there is no clear favorite and none of the languages is able to meet all requirements raised by data practitioners. To support the work, a comprehensive, community-driven database has been created where case studies, use cases, requirements and solutions are collected. Based on this database, we have hitherto published 81 types of constraints that are required by various stakeholders for data applications. We are using this collection of constraint types to gain a better understanding of the expressiveness of existing solutions and gaps that still need to be filled. Regarding the implementation of constraint languages, we have already proposed to use high-level languages to describe the constraints, but map them to SPARQL queries in order to execute the actual validation; we have demonstrated this approach for the Web Ontology Language in its current version 2 and Description Set Profiles. In this paper, we generalize from the experience of implementing OWL 2 and DSP by introducing an abstraction layer that is able to describe constraints of any constraint type in a way that mappings from high-level constraint languages to this intermediate representation can be created more or less straight-forwardly. We demonstrate that using another layer on top of SPARQL helps to implement validation consistently accross constraint languages, simplifies the actual implementation of new languages, and supports the transformation of semantically equivalent constraints across constraint languages.

Keywords: RDF validation; RDF constraints; RDF constraint types, RDF validation requirements; Linked Data; Semantic Web

1. Introduction

The proper validation of RDF data according to constraints is a common requirement of data practitioners. Among the reasons for the success of XML is the possibility to formulate fine-grained constraints to be met by the data and to validate the data according to these constraints using powerful systems like DTD, XML Schema, RELAX NG, or Schematron.

In 2013, the W3C organized the *RDF Validation Workshop*[1] where experts from industry, government, and academia discussed first RDF validation use cases. In 2014, two working groups on RDF validation were established: the *W3C RDF Data Shapes Working Group*[2] and the *DCMI RDF Application Profiles Task Group*.[3] We collected the findings of these working groups and initiated a database of RDF validation requirements[4] with the intention to collaboratively collect case studies, use cases, requirements, and solutions in a comprehensive and structured way (Bosch & Eckert, 2014a). Based on our work in the DCMI and in cooperation with the W3C

[1] http://www.w3.org/2012/12/rdf-val/
[2] http://www.w3.org/2014/rds/charter
[3] http://wiki.dublincore.org/index.php/RDF-Application-Profiles
[4] Online available at: http://purl.org/net/rdf-validation

https://doi.org/10.23106/dcmi.952137055

working group, we identified by today 81 constraint types, where each type corresponds to a specific requirement in the database. In a technical report, we explain each constraint type in detail and give examples for each represented by different constraint languages (Bosch, Nolle, Acar, & Eckert, 2015).

Various constraint languages exist or are being developed that support more or less of these constraint types. For our work, we focus on the following four as the ones that are most popular among data practitioners, often mentioned on mailing lists and/or being candidates or prototypes for the upcoming W3C recommendation: *Description Set Profiles (DSP)*,[5] *Resource Shapes (ReSh)*,[6] *Shape Expressions (ShEx)*,[7] and the *Web Ontology Language (OWL)*.[8] Despite the fact that OWL is arguably not a constraint language, it is widely used in practice as such under the closed-world and unique name assumptions.

With its direct support of validation via SPARQL, the *SPARQL Inferencing Notation (SPIN)*[9] is also very popular to formulate and check constraints (Fürber & Hepp, 2010). We consider SPIN as a low-level language in contrast to the other constraint languages where specific language constructs exist to define constraints in a declarative and in comparison more intuitive way – although SPARQL aficionados might object particularly to the latter point.

The power of SPIN is shown in Table 1, where we list the fraction (and absolute numbers in brackets) of how many constraint types each of these languages supports (Bosch et al., 2015). We further see that OWL 2 is currently the most expressive high-level constraint language, at least according to the pure number of constraint types supported. This does not preclude that other constraint languages are better suited for certain applications, either because they support some types that are not supported by OWL or because the constraint representation is more appealing to the data practitioners – producers as well as consumers who again might have different needs and preferences.

TABLE 1: Constraint Type Specific Expressivity of Constraint Languages

DSP	ReSh	ShEx	OWL 2	SPIN
17.3 (14)	25.9 (21)	29.6 (24)	67.9 (55)	100.0 (81)

We formerly demonstrated that a high-level constraint language like OWL 2 and DSP can be implemented by mapping the language to SPIN using SPARQL CONSTRUCT queries (Bosch & Eckert, 2014b). We provide a validation environment where own mappings from arbitrary constraint languages can be provided and tested.[10] The only limitations are that the constraints have to be expressed in RDF and that the constraint language is expressible in SPARQL.

The constraint type *minimum qualified cardinality restrictions* which corresponds to the requirement *R-75*[11] can be instantiated to formulate the constraint that publications must have at least one author which must be a person. This constraint can be expressed as follows using different constraint languages:

[5] http://dublincore.org/documents/2008/03/31/dc-dsp/
[6] http://www.w3.org/Submission/2014/SUBM-shapes-20140211/
[7] http://www.w3.org/Submission/2014/SUBM-shex-primer-20140602/
[8] http://www.w3.org/TR/owl2-syntax/
[9] http://spinrdf.org/
[10] Online available at: http://purl.org/net/rdfval-demo, source code online available at: https://github.com/boschthomas/rdf-validator.
[11] Requirements are identified in the database by an R and a number, additionally an alphanumeric identifier is provided, in this case *R-75-MINIMUM-QUALIFIED-CARDINALITY-ON-PROPERTIES*. Online at: http://lelystad.informatik.uni-mannheim.de/rdf-validation/?q=node/82

https://doi.org/10.23106/dcmi.952137055

```
OWL 2: Publication a owl:Restriction ;
          owl:minQualifiedCardinality 1 ;
          owl:onProperty author ;
          owl:onClass Person .

ShEx: Publication { author @Person{1, } }

ReSh: Publication a rs:ResourceShape ; rs:property [
          rs:propertyDefinition author ;
          rs:valueShape Person ;
          rs:occurs rs:One-or-many ; ] .

DSP: [ dsp:resourceClass Publication ; dsp:statementTemplate [
          dsp:minOccur 1 ;
          dsp:property author ;
          dsp:nonLiteralConstraint [ dsp:valueClass Person ] ] ] .

SPIN: CONSTRUCT { [ a spin:ConstraintViolation ... . ] } WHERE {
          ?this
              a ?C1 ;
              ?p ?o .
          BIND ( qualifiedCardinality( ?this, ?p, ?C2 ) AS ?c ) .
          BIND( STRDT ( STR ( ?c ), xsd:nonNegativeInteger ) AS ?cardinality ) .
          FILTER ( ?cardinality < 1 ) .
          FILTER ( ?C1 = Publication ) .
          FILTER ( ?C2 = Person ) .
          FILTER ( ?p = author ) . }

SPIN function qualifiedCardinality:
SELECT ( COUNT ( ?arg1 ) AS ?c ) WHERE { ?arg1 ?arg2 ?o . ?o a ?arg3 . }
```

Note that the SPIN representation of the constraint is *not* a SPIN mapping to implement the constraint, but a direct expression of the constraint using a SPARQL CONSTRUCT query that creates a spin:ConstraintViolation if the constraint is violated.

It can be seen that the higher-level constraint languages are comparatively similar, there seems to be a pattern, a common way to express this type of constraint. Therefore, a mapping from a high-level language to another high-level language would be considerably easier. Unfortunately, there is not (yet) a high-level language that supports all constraint types.

The creation of mappings of constraint languages to SPIN to implement their validation is in many cases not straight-forward and requires profound knowledge of SPARQL, as the following example demonstrates. In this example, the validation of the *minimum qualified cardinality restrictions* constraint type is implemented for DSP:

```
CONSTRUCT {
    _:constraintViolation
        a spin:ConstraintViolation ;
        rdfs:label ?violationMessage ;
        spin:violationRoot ?this ;
        spin:violationPath ?property ;
        spin:violationSource ?violationSource . }
WHERE {
    ?this a ?resourceClass .
    ?descriptionTemplate
        dsp:resourceClass ?resourceClass ;
        dsp:statementTemplate ?statementTemplate .
    ?statementTemplate
        dsp:minOccur ?minimum ;
        dsp:property ?property ;
        dsp:nonLiteralConstraint ?nonLiteralConstraint .
    ?nonLiteralConstraint dsp:valueClass ?valueClass .
    BIND ( qualifiedCardinality( ?this, ?property, ?valueClass ) AS ?cardinality ) .
    FILTER ( ?cardinality < ?minimum ) . }
```

The SPIN mappings for OWL 2 and DSP are rather complicated and can be found in

https://doi.org/10.23106/dcmi.952137055

the mappings provided by us.[12]

In this paper, we build on the experience gained from mapping several constraint languages to SPIN and from the analysis of the identified constraint types to create an intermediate layer, a framework that is able to describe the mechanics of all constraint types and that can be used to map high-level languages more easily.

2. Motivation

Even with an upcoming W3C recommendation, it can be expected that several constraint languages will be used in practice in future – consider the situation in the XML world, where a standardized schema language was available from the beginning and yet additional ways to formulate and check constraints have been created. Therefore, semantically equivalent constraints represented in different languages will exist. This raises two questions:

1. How can we ensure that two semantically equivalent constraints are actually validated consistently?
2. How can we support the transformation of semantically equivalent constraints from one constraint language to another?

Consistent implementation. Even though SPIN provides a convenient way to represent constraints and to validate data according to these constraints, the implementation of a high-level constraint language still requires a tedious mapping to SPIN with a certain degree of freedom as to how a constraint violation is actually represented and how exactly the violation of the constraint is checked. Our framework therefore provides a common ground that is solely based on the abstract definitions of the constraint types, as identified in our database. By providing a SPIN mapping for each constraint type,[13] it is ensured that the details of the SPIN implementation are consistent irrespective of the constraint language and that the validation leads always to exactly the same results.

Constraint transformation. Consistent implementations of constraint languages provide some advantage, but it could be argued that they are not important enough to justify the additional layer. The situation, however, is different when transformations from one constraint language to another are desired, i.e., to transform a *specific constraint* sc_α of any constraint type expressed by language α into a semantically equivalent *specific constraint* sc_β of the same constraint type represented by any other language β. By defining mappings between equivalent *specific constraints* and the corresponding *generic constraint* (*gc*) we are able to convert them automatically:

$$gc = m_\alpha (sc_\alpha)$$
$$sc_\beta = m'_\beta (gc)$$

Thereby, we do not need to define mappings for each constraint type and each possible combination of constraint languages. Assuming that we are able to express a single constraint type like *minimum qualified cardinality restrictions* within 10 languages, $n \cdot n-1 = 90$ mappings would be needed – as mappings generally are not invertible. With an intermediate generic

[12] OWL 2 mapping online available at: `https://github.com/boschthomas/rdf-validation/blob/b6a275fb5d71a92ae33d3b6aadd5f447351214b7/SPIN/OWL2_SPIN-Mapping.ttl`; DSP mapping online available at: `https://github.com/boschthomas/rdf-validation/blob/b6a275fb5d71a92ae33d3b6aadd5f447351214b7/SPIN/DSP_SPIN-Mapping.ttl#L4665`

[13] RDF-CV to SPIN online available at: `https://github.com/boschthomas/RDF-CV-2-SPIN`

https://doi.org/10.23106/dcmi.952137055

representation of constraints, on the other side, we only need to define for each constraint type 2n = 20 mappings – where 10 mappings should already exist if we have an implementation in our framework. To summarize, if language developers are willing to provide two mappings – forward (m) and backward (m´) – to our framework for each supported constraint type, we not only would get the consistent implementation of all languages, it would also be possible to transform semantically equivalent constraints into all constraint languages.

3. Towards a Framework

When we fully implemented OWL 2 and DSP and to some extend other constraint languages using SPARQL as intermediate language (Bosch & Eckert, 2014b), we found that many mappings actually resemble each other; particularly the mappings of the same constraint type in different languages, but also the mappings of different constraint types, though the latter only on a very superficial, structural level. The basic idea of our framework is very simple: we aim at reducing the representation of constraints to the absolute minimum that has to be provided in a mapping to SPIN to implement the validation for constraint types. Consider again our example

```
SPIN: CONSTRUCT { [ a spin:ConstraintViolation ... . ] } WHERE {
          ?this
              a ?C1 ;
              ?p ?o .
          BIND ( qualifiedCardinality( ?this, ?p, ?C2 ) AS ?c ) .
          BIND( STRDT ( STR ( ?c ), xsd:nonNegativeInteger ) AS ?cardinality ) .
          FILTER ( ?cardinality < 1 ) .
          FILTER ( ?C1 = Publication ) .
          FILTER ( ?C2 = Person ) .
          FILTER ( ?p = author ) . }

SPIN function qualifiedCardinality:
SELECT ( COUNT ( ?arg1 ) AS ?c ) WHERE { ?arg1 ?arg2 ?o . ?o a ?arg3 . }
```

from above for the SPIN representation of a constraint of the type *minimum qualified cardinality restrictions*:

However this SPIN code looks like, all we have to provide to make it work is the desired minimum cardinality (?cardinality), the property to be constrained (?p), the class whose individuals must hold for the constraint (?C1), and the class for which the property should be

```
OWL 2: Publication a owl:Restriction ;
          owl:minQualifiedCardinality 1 ;
          owl:onProperty author ;
          owl:onClass Person .

ShEx: Publication { author @Person{1, } }

ReSh: Publication a rs:ResourceShape ; rs:property [
          rs:propertyDefinition author ;
          rs:valueShape Person ;
          rs:occurs rs:One-or-many ; ] .

DSP: [ dsp:resourceClass Publication ; dsp:statementTemplate [
          dsp:minOccur 1 ;
          dsp:property author ;
          dsp:nonLiteralConstraint [ dsp:valueClass Person ] ] ] .
```

constrained (?C2). All other variables are bound internally. So we could reduce the effort of the mapping by simply providing these four values, which are readily available in all representations of this constraint type:

In further investigation of all kind of constraints and particularly the list of constraint types, we aimed at identifying the building blocks of such constraints to come up with a concise representation of every constraint type.

https://doi.org/10.23106/dcmi.952137055

3.1. Building Blocks

At the core, we use a very simple conceptual model for constraints (see Figure 1), using a small lightweight vocabulary called *RDF Constraints Vocabulary (RDF-CV)*. [14]

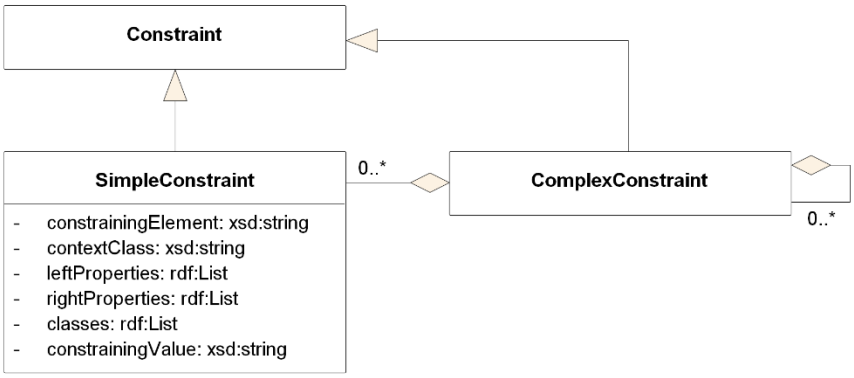

FIG. 1. *RDF Constraints Vocabulary (RDF-CV)* Conceptual Model

RDF constraints are either simple constraints or complex constraints. *Simple constraints* denotes the set of atomic constraints with respect to a single constraining element – we will come to the notion of a constraining element in a second. In contrast, there are *complex constraints*, i.e., the set of constraints which are created out of simple and/or other complex constraints. This structure therefore allows to build complex constraints out of other (simple or complex) constraints. Regarding our database of constraint types, 60% of the constraint types are used to instantiate simple constraints and 26% complex constraints. Constraints of additional 14% of the constraint types are complex constraints as well which can be simplified and therefore formulated as simple constraints if additional constraining elements are introduced to cover them.

The properties describing a simple constraint are very structural, i.e., the properties describe the structure of constraints. The central property is the *constraining element* which refers to one of 103 constraining elements described in our technical report (Bosch et al., 2015). Constraining elements are for example taken from Description Logics, another concrete example would be the SPARQL function REGEX where a regular expression is checked against some property value. In most cases, constraining elements directly correspond to a constraint type, sometimes (as for REGEX) they are shared by several constraint types. Complex constraints again need several constraining elements to be expressed.

Irrespective of and additional to the constraining element, there are properties to describe the actual constraint, they can also be seen as parameters for the constraining element. The *context class* limits the constraint to individuals of a specific class. Depending on the constraining elements, a list of *classes* can be provided, for example to determine the valid classes for a value or to define a class intersection to be used in a constraint. *leftProperties* and *rightProperties* are lists usually containing properties the constraint is applied to. A typical example for a constraint type with a right hand side list of properties would be *literal value comparison (R-43)*, where constraints like `birthDate < deathDate` can be expressed. Finally, the *constraining value* contains a literal value to be checked against; for instance in the case of the REGEX element, it contains the regular expression to be evaluated.

This simple structure plus the constraining elements form the building blocks of our proposed framework. In the technical report (Bosch et al., 2015), we list for every constraint type its representation in our framework which not only shows that constraints of any constraint type can

[14] Formal specification and HTML documentation online available at: `https://github.com/boschthomas/RDF-Constraints-Vocabulary`

https://doi.org/10.23106/dcmi.952137055

indeed be described generically in this way, but which also forms the starting point for any mappings using this framework.

Formal approach and semantics. A cornerstone of the framework is the generic representation of a constraint, which can often be done using Description Logics. For example the *minimum qualified cardinality restriction* can be expressed as `Publication ⊑ ≥1 author.Person`. This way, the knowledge representation formalism *Description Logics (DL)* (Krötzsch, Simancík, & Horrocks, 2012; Baader, Calvanese, McGuinness, Nardi, & Patel-Schneider, 2003; Baader & Nutt, 2003) with its well-studied theoretical properties provides the foundational basis for the framework.

It turned out that 64% of the 81 constraint types are actually expressible in DL. Only for the remaining 36%, other means, i.e., other constraining elements, had to be identified. This is not surprising if we consider that OWL is based on DL. When we talk about using DL to represent constraints, we have to establish once more that the semantics of OWL and DL differ from the semantics of constraint languages regarding the open world assumption (OWA) and the non-unique name assumption (nUNA). Both are usually assumed when dealing with OWL or DL, whereas validation usually assumes a closed world (CWA) and unique naming (UNA), i.e., if a desired property is missing, this leads to a violation and if two resources are named differently, they are assumed to be different resources.

We won't get into details about these assumptions here, but it has to be noted that the applied semantics have to be defined if validation is performed, as the results would differ under different semantics. Precisely, we found that for 56.8% of the constraint types validation results differ if the CWA or the OWA is assumed and for 66.6% of the constraint types validation results are different in case the UNA or the nUNA is assumed (Bosch et al., 2015).

For the purpose of a consistent implementation and transformation of constraints, constraints are considered *semantically equivalent* if they detect the same set of violations regardless of RDF data, which means whenever the constraints are applied to any RDF data they point out the same violations.

3.2. Simple Constraints

In this and the following section, we provide examples for the representation of constraint types within the framework.

The *minimum qualified cardinality restriction (R-75)* `Publication ⊑ ≥1 author.Person`, which restricts publications to have at least one author which must be a person, is an example of a simple constraint on *author* which holds for all individuals of the class *Publication*. Table 2 displays how the simple constraint is generically represented using the RDF-CV.

TABLE 2: Minimum Qualified Cardinality Restriction as Property Constraint

context class	left property list	right p. list	classes	constraining element	c. value
Publication	author	-	Person	≥	1

The *constraining element* is an intuitive term which indicates the actual type of constraint. For the majority of the constraint types, there is exactly one constraining element, for instance *property domain (R-25, R-26)* restricts domains of properties and there is only one constraining element with exactly the same identifier *property domain*. Some constraint types, however, need several constraining elements to be expressed, for instance *language tag cardinality (R-48, R-49)* is used to restrict data properties to have a minimum, maximum, or exact number of relationships to literals with selected language tags. Thus, three constraining elements are needed to express each possible constraint of that constraint type. This example also illustrates that the granularity of the constraint types varies and certainly often is debatable. Keep in mind that they correspond

https://doi.org/10.23106/dcmi.952137055

to requirements as identified by the working groups. The constraining elements, as in this example, are closer to atomic elements of constraints.

If constraint types are expressible in DL, associated constraining elements are formally based on DL constructs like concept and role constructors ($\sqsubseteq$, $\equiv$, $\sqcap$, $\sqcup$, $\neg$, $\exists$, $\forall$, $\geq$, $\leq$), equality ($=$), and inequality ($\neq$). In case constraint types cannot be expressed in DL such as *data property facets (R-46)* or *literal pattern matching (R-44)*, we reuse widely known terms from SPARQL (e.g., REGEX) or XML Schema constraining facets (e.g., *xsd:minInclusive*) as constraining elements. We provide a complete list of all 103 constraining elements which can be used to express constraints of any constraint type (Bosch et al., 2015).

Additional to the constraining element, there are properties of simple constraints which can be seen as parameters for the constraining element. In some cases, a simple constraint is only complete when a *constraining value* is stated in conjunction with the constraining element. Depending on the constraining element, a list of *classes* can be provided, for example to determine the valid classes for a value. The constraining element of the constraint `Publication` $\sqsubseteq$ `≥1 author.Person`, e.g., is $\geq$, the constraining value is *1*, and the list of classes includes the class *Person* which restricts the objects of the property *author* to be persons. The assignment of properties to the left or right property lists depends on the constraining element.

Object property paths (R-55) ensure that if an individual *x* is connected by a sequence of object properties with an individual *y*, then *x* is also related to *y* by a particular object property. As *Stephen-Hawking* is the author of the book *A-Brief-History-Of-Time* whose genre is *Popular-Science*, the object property path `authorOf ∘ genre` $\sqsubseteq$ `authorOfGenre` infers that *Stephen-Hawking* is an author of the genre *Popular-Science*. Thus, when representing the constraint using the RDF-CV (see Table 3), the properties *authorOf* and *genre* are placed on the left side of the constraining element *property path* and the property *authorOfGenre* on its right side. The *context class* limits the constraint to individuals of a specific class. A context class may be an rdfs:Class, an owl:Class (as sub-class of rdfs:Class), or an rdfs:Datatype which is both an instance of and a sub-class of rdfs:Class. As the *property path* constraint holds for all individuals within the data, the context class is set to the *DL top concept* $\top$ which stands for the super-class of all possible classes.

TABLE 3: Object Property Paths as Property Constraint

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	authorOf, genre	authorOfGenre	$\top$	property path	-

Constraints of 36% of the constraint types are not expressible in DL but can still be described using the RDF-CV such as constraints of the type *literal pattern matching (R-44)* which restrict literals to match given patterns. The *universal quantification (R-91)* `Book` $\sqsubseteq$ $\forall$ `identifier.ISBN` ensures that books can only have valid *ISBN* identifiers, i.e., strings that match a given regular expression.

Even though constraints of the type *literal pattern matching* cannot be expressed in DL, OWL

```
ISBN a RDFS:Datatype ; owl:equivalentClass [ a RDFS:Datatype ;
    owl:onDatatype xsd:string ;
    owl:withRestrictions ([ xsd:pattern "^\d{9}[\d|X]$" ])] .
```

2 can be used to formulate this constraint:

The first OWL 2 axiom explicitly declares *ISBN* to be a datatype. The second OWL 2 axiom defines *ISBN* as an abbreviation for a datatype restriction on *xsd:string*. The datatype *ISBN* can be used just like any other datatype like in the universal quantification above.

https://doi.org/10.23106/dcmi.952137055

Table 4 presents (1) how the not in DL expressible literal pattern matching constraint and (2) how the in DL expressible universal quantification are both represented using the RDF-CV. Thereby, the context class *ISBN*, whose instances must satisfy the literal pattern matching constraint, is reused within the list of classes the universal quantification refers to. The literal pattern matching constraint type introduces the constraining element *REGEX* whose validation has to be implemented once like for any other constraining element.

TABLE 4: Simple Constraints which are not Expressible in DL

context class	left p. list	right p. list	classes	c. element	c. value
ISBN	-	-	xsd:string	REGEX	`"^\d{9}[\d|X]$"`
Book	identifier	-	ISBN	universal quantification	-

3.3. Complex Constraints

Complex constraints of the constraint type *context-specific exclusive or of property groups (R-13)* restrict individuals of given classes to have all properties of exactly one of multiple mutually exclusive property groups. Publications, e.g., are either identified by an ISBN and a title (for

```
Publication {
    ( isbn string , title string ) |
    ( issn string , title string ) }
```

books) or by an ISSN and a title (for periodical publications), but it should not be possible to assign both identifiers to a given publication. This complex constraint is expressible in ShEx:

If *The-Great-Gatsby* is a publication with an ISBN and a title without an ISSN, *The-Great-Gatsby* is considered as a valid publication. This complex constraint is generically expressible in DL:

$$\mathbf{Publication} \sqsubseteq (\neg E \sqcap F) \sqcup (E \sqcap \neg F) \;,\; E \equiv A \sqcap B \;,\; F \equiv C \sqcap D$$

$$A \sqsubseteq \geq 1\ \mathtt{isbn.string} \sqcap \leq 1\ \mathtt{isbn.string} \;,\; B \sqsubseteq \geq 1\ \mathtt{title.string} \sqcap \leq 1\ \mathtt{title.string}$$

$$C \sqsubseteq \geq 1\ \mathtt{issn.string} \sqcap \leq 1\ \mathtt{issn.string} \;,\; D \sqsubseteq \geq 1\ \mathtt{title.string} \sqcap \leq 1\ \mathtt{title.string}$$

The DL statements demonstrate that the complex constraint is composed of many other complex constraints (*minimum (R-75) and maximum qualified cardinality restrictions (R-76)*) and simple constraints (*intersection (R-15/16)*, *disjunction (R-17/18)*, and *negation (R-19/20)*). Constraints of almost 14% of the constraint types are complex constraints which can be simplified and therefore formulated as simple constraints when using them in terms of syntactic sugar. As *exact (un)qualified cardinality restrictions (R-74/80) (=n)* and *exclusive or of property groups (R-13)* are constraint types of frequently used complex constraints, we propose to simplify them in form of simple constraints. As a consequence, the *context-specific exclusive or of property groups* complex constraint is represented as a generic constraint by means of the RDF-CV more intuitively and concisely (see Table 5).

TABLE 5: Simplified Complex Constraints

context class	left p. list	right p. list	classes	c. element	c. value
Publication	-	-	E, F	exclusive or	-
E	-	-	A, B	intersection	-
F	-	-	C, D	intersection	-

https://doi.org/10.23106/dcmi.952137055

A	isbn	-	string	=	1
B	title	-	string	=	1
C	issn	-	string	=	1
D	title	-	string	=	1

The *primary key properties (R-226)* constraint type is often useful to declare a given (datatype) property as the primary key of a class, so that a system can enforce uniqueness. Books, e.g., are uniquely identified by their ISBN, i.e., the property *isbn* is inverse functional (`funct isbn⁻`) which can be represented using the RDF-CV in form of a complex constraint consisting of two simple constraints (see Table 6). The meaning of these simple constraints is that ISBN identifiers can only have `isbn⁻` relations to at most one distinct book.

TABLE 6: Primary Key Properties as Complex Constraints

context class	left p. list	right p. list	classes	c. element	c. value
⊤	isbn⁻	isbn	-	inverse property	-
Book	isbn⁻	-	-	≤	1

Keys, however, are even more general, i.e., a generalization of inverse functional properties (Schneider, 2009). A key can be a datatype, an object property, or a chain of properties. For these generalization purposes, as there are different sorts of keys, and as keys can lead to undecidability, DL is extended with a special construct *keyfor* (Lutz, Areces, Horrocks, & Sattler, 2005). When using keyfor (`isbn keyfor Book`), the complex constraint can be simplified and thus

```
[   a rdfcv:SimpleConstraint ;
    rdfcv:contextClass Book ;
    rdfcv:leftProperties ( isbn ) ;
    rdfcv:constrainingElement "primary key" ] .
```

formulated as a simple constraint which looks like the following in concrete RDF turtle syntax:

Complex constraints of frequently used constraint types which correspond to DL axioms like *transitivity, symmetry, asymmetry, reflexivity* and *irreflexivity* can also be simplified in form of simple constraints. Although these DL axioms are expressible by basic DL features, they can also be used in terms of syntactic sugar.

Constraints of the *irreflexive object properties (R-60)* constraint type ensure that no individual is connected by a given object property to itself (Krötzsch et al., 2012). With the irreflexive object property constraint `⊤ ⊑ ¬∃authorOf.Self` , e.g., one can state that individuals cannot be authors of themselves. When represented using the RDF-CV, the complex constraint aggregates three simple constraints (see Table 7).

TABLE 7: Irreflexive Object Properties as Complex Constraints

context class	left p. list	right p. list	classes	c. element	c. value
∃ authorOf.Self	authorOf	-	Self	existential quantification	-
¬∃ authorOf.Self	-	-	∃ authorOf.Self	negation	-
⊤	-	-	⊤, ¬∃ authorOf.Self	sub-class	-

https://doi.org/10.23106/dcmi.952137055

When using the *irreflexive object property* constraint in terms of syntactic sugar, the complex constraint can be expressed more concisely in form of a simple property constraint with exactly the same semantics (see Table 8):

TABLE 8: Irreflexive Object Properties as Simple Constraints

context class	left p. list	right p. list	classes	c. element	c. value
⊤	authorOf	-	-	irreflexive property	-

3.4. Mapping Implementation

Using the framework for the implementation of a constraint language is straight-forward. For each language construct, the corresponding constraint type has to be identified. Again we use the constraint `Publication ⊑ ≥1 author.Person` of the type *minimum qualified cardinality restrictions (R-75)* which is supported in OWL 2:

```
:Publication
    a owl:Restriction ;
    owl:minQualifiedCardinality 1 ;
    owl:onProperty :author ;
    owl:onClass :Person .
```

From Table 2, we know the representation in our framework, which corresponds to the following RDF representation using the RDF-CV:

```
[   a rdfcv:SimpleConstraint ;
    rdfcv:contextClass :Publication ;
    rdfcv:leftProperties ( :author ) ;
    rdfcv:classes ( :Person ) ;
    rdfcv:constrainingElement "minimum qualified cardinality restriction" ;
    rdfcv:constrainingValue 1 ] .
```

The mapping simply constructs this generic representation out of the specific OWL 2 representation using a SPARQL CONSTRUCT query:

```
owl:Thing
    spin:rule [ a sp:Construct ; sp:text """
        CONSTRUCT {
            :minimum-qualified-cardinality-restrictions
                a rdfcv:SimpleConstraint ;
                rdfcv:contextClass ?this ;
                rdfcv:leftProperties :leftProperties ;
                rdfcv:classes :classes ;
                rdfcv:constrainingElement "minimum qualified cardinality restriction"
;
                rdfcv:constrainingValue ?cv .
            :leftProperties
                rdf:first ?lp1 ;
                rdf:rest rdf:nil .
            :classes
                rdf:first ?c1 ;
                rdf:rest rdf:nil . }
        WHERE {
            ?this
                a owl:Restriction ;
                owl:minQualifiedCardinality ?cv ;
                owl:onProperty ?lp1 ;
                owl:onClass ?c1 . } """ ; ] .
```

The SPIN engine is used to execute the mapping, the property *spin:rule* links an rdfs:Class with SPARQL CONSTRUCT queries. Each query defines an inference rule that is applied to all instances of the associated class and its subclasses. The inference rule defines how additional

https://doi.org/10.23106/dcmi.952137055

triples can be inferred from what is stated in the WHERE clause. For each binding of the pattern in the WHERE clause of the rule, the triple templates from the CONSTRUCT clause are instantiated and added as inferred triples to the underlying model. At query execution time, the SPARQL variable *?this* is bound to the current instance of the class. As each resource per default is assigned to the class *owl:Thing*, this inference rule is evaluated for each subject of the input RDF graph.

The framework and therefore the constraint types are implemented in exactly the same way by providing other SPIN mappings which encompass the SPIN/SPARQL queries that validate constraints and produce constraint violation messages if a constraint is violated, as described in our previous paper about the DSP implementation (Bosch & Eckert, 2014b).[15]

3.5. Constraint Transformation

As stated in Section 2, we see a huge potential in the possibility to transform semantically equivalent constraints from one high-level constraint language to another via the RDF-CV representation, to avoid that every possible combination of constraint languages has to be mapped

```
owl:Thing
    spin:rule [ a sp:Construct ; sp:text """
        CONSTRUCT {
            ?cc
                a owl:Restriction ;
                owl:minQualifiedCardinality ?cv ;
                owl:onProperty ?lp1 ;
                owl:onClass ?c1 . }
        WHERE {
            ?this
                a rdfcv:SimpleConstraint ;
                rdfcv:contextClass ?cc ;
                rdfcv:leftProperties ?leftProperties ;
                rdfcv:classes ?classes ;
                rdfcv:constrainingElement "minimum qualified cardinality restriction"
;
                rdfcv:constrainingValue ?cv .
            ?leftProperties
                rdf:first ?lp1 ;
                rdf:rest rdf:nil .
            ?classes
                rdf:first ?c1 ;
                rdf:rest rdf:nil . } """ ; ] .
```

separately. The following SPIN inference rule exemplifies this approach and provides a mapping from RDF-CV back to the OWL 2 constraint of the type *minimum qualified cardinality restrictions*:

It can be seen that the mapping is quite similar to the first mapping and basically simply switches the CONSTRUCT and WHERE part of the query, with slight adjustment in the structure of the variables. Potentially an even simpler representation for the mapping could be found that would enable the creation of forward and backward mappings out of it. We didn't investigate this further, though, and it is not yet clear if there can be cases where the backward mapping is more different.

4. Related Work

In this section, we present current languages for RDF constraint formulation and RDF data validation. SPIN, SPARQL, OWL 2, ShEx, ReSh, and DSP are the six most promising and

[15] At the time of this writing, not all mappings for the constraint types are implemented, but of course the implementations can be complemented and adapted to own requirements, as needed. The most recent implementation can be found here: https://github.com/boschthomas/rdf-validation/blob/master/SPIN/RDF-CV-2-SPIN.ttl

https://doi.org/10.23106/dcmi.952137055

mostly used constraint languages. In addition, the W3C Data Shapes Working Group currently develops SHACL, an RDF vocabulary for describing RDF graph structures.

The *SPARQL Query Language for RDF* (Harris & Seaborne, 2013) is generally seen as the method of choice to validate RDF data according to certain constraints (Fürber & Hepp, 2010), although, it is not ideal for their formulation. In contrast, high-level constraint languages are comparatively easy to understand and constraints can be formulated more concisely. Declarative languages may be placed on top of SPARQL and SPIN when using them as implementation languages. The *SPARQL Inferencing Notation (SPIN)*[16] (Knublauch, Hendler, & Idehen, 2011) provides a vocabulary to represent SPARQL queries as RDF triples and uses SPARQL to specify logical constraints and inference rules (Fürber & Hepp, 2010). Kontokostas et al. define 17 data quality integrity constraints represented as SPARQL query templates called *Data Quality Test Patterns (DQTP)* (Kontokostas et al., 2014).

The *Web Ontology Language (OWL)* (Hitzler, Krötzsch, Parsia, Patel-Schneider, & Rudolph, 2012) formally specifies the intended semantics of conceptual models about data and therefore enables software to understand data. OWL has become a popular standard for data representation, data exchange, and data integration of heterogeneous data sources. Besides that, the retrieval of data benefits from semantic knowledge specified using OWL. In combination with the OWL-based *Semantic Web Rule Language (SWRL)* (Horrocks et al., 2004), OWL provides facilities for developing very powerful reasoning services. Reasoning on RDF data enables to derive implicit data out of explicitly stated data. OWL is based on formal logic and on the subject-predicate-object triples from RDF. OWL is actually a description logic with underlying formal semantics which allows one to assign truth values to syntactic expressions. OWL specifies semantic information about specific domains, describes relations between domain classes, and thus allows the sharing of conceptualizations.

Because of the design of OWL for reasoning, there are claims that OWL cannot be used for validation. In practice, however, OWL is well-spread and RDFS/OWL constructs are widely used to tell people and applications about how valid instances should look like. In general, RDF documents follow the syntactic structure and the semantics of RDFS/OWL ontologies which could therefore not only be used for reasoning but also for validation.

Stardog Integrity Constraint Validation (ICV) and the *Pellet Integrity Constraint Validator (ICV)* use OWL 2 constructs to formulate constraints. The Pellet ICV[17] is a proof-of-concept extension for the OWL 2 DL reasoner *Pellet* (Sirin, Parsia, Grau, Kalyanpur, & Katz, 2007). Stardog ICV[18] validates RDF data stored in a Stardog database according to constraints which may be written in SPARQL, OWL 2, or SWRL (Horrocks et al., 2004).

Shape Expressions (ShEx) (Prud'hommeaux, 2014; Solbrig & Prud'hommeaux, 2014; Prud'hommeaux, Labra Gayo, & Solbrig, 2014; Boneva et al., 2014) specifies a language whose syntax and semantics are similar to regular expressions. ShEx associate RDF graphs with labeled patterns called *shapes* which are used to express formal constraints on the content of RDF graphs. *Resource Shapes (ReSh)* (A. Ryman, 2014) defines its own vocabulary for specifying shapes of RDF resources. Ryman, Hors, and Speicher define *shape* as a description of the set of triples a resource is expected to contain and as a description of the integrity constraints those triples are required to satisfy (A. G. Ryman, Hors, & Speicher, 2013).

The *Dublin Core Application Profile (DCAP)* and *Bibframe Profiles* are approaches to specify profiles for application-specific purposes. The term *profile* is widely used to refer to a document that describes how standards or specifications are deployed to support the requirements of a particular application, function, community, or context. In the metadata community, the term *application profile* has been applied to describe the tailoring of standards for specific

[16] http://spinrdf.org
[17] http://clarkparsia.com/pellet/icv
[18] http://docs.stardog.com/#_validating_constraints

https://doi.org/10.23106/dcmi.952137055

applications. A *Dublin Core Application Profile (DCAP)* (Coyle & Baker, 2009) defines metadata records which meet specific application needs while providing semantic interoperability with other applications on the basis of globally defined vocabularies and models. The *Singapore Framework for Dublin Core Application Profiles* (Nilsson, Baker, & Johnston, 2008) is a framework for designing metadata and for defining DCAPs. The framework comprises descriptive components that are necessary or useful for documenting DCAPs.

The *DCMI Abstract Model* (Powell, Nilsson, Naeve, Johnston, & Baker, 2007) is required for formalizing a notion of machine-processable application profiles. It specifies an abstract model for Dublin Core metadata which is independent of any particular encoding syntax. Its primary purpose is to specify the components used in Dublin Core metadata. Nilsson et al. (Nilsson, Powel, Johnston, & Naeve, 2008) depict how the constructs of the DCMI Abstract Model are represented using the abstract syntax of the RDF model. A *Description Set Profile (DSP)* (Nilsson, 2008) is a generic constraint language which is used to formally specify structural constraints on sets of resource descriptions within an application profile. DSP constrains resources that may be described by descriptions in a description set, the properties that may be used, and the values properties may point to. *BIBFRAME*[19] (Kroeger, 2013; Godby, Carol Jean and Denenberg, Ray, 2015; Miller, Eric and Ogbuji, Uche and Mueller, Victoria and MacDougall, Kathy, 2012) is the result of the *Bibliographic Framework Initiative* and defines a vocabulary (Library of Congress, 2014a, 2014c) which has a strong overlap with DSP. *BIBFRAME Profiles* (Library of Congress, 2014b) are essentially identical to DCAPs.

Schemarama[20] is a validation technique for specifying the types of sub-graphs you want to have connected to a particular set of nodes in an RDF Graph. Schemarama allows to check that RDF data has required properties. Schemarama is based on Schematron (ISO/IEC, 2006), an XML schema and XML structure validation language which works by finding tree patterns within an XML document. Schemarama is also based on the *Squish RDF Query language* (Miller, 2001), an SQL-like query language for RDF, instead of SPARQL.

In addition to the formulation of constraints, SPIN (open source API), Stardog ICV (as part of the Stardog RDF database), DQTP (tests), Pellet ICV (extension of Pellet OWL 2 DL reasoner) and ShEx offer executable validation systems using SPARQL as implementation language.

The W3C Data Shapes Working Group currently develops *SHACL* (Knublauch, 2015; Boneva & Prud'hommeaux, 2015; Prud'hommeaux, 2015), the *Shapes Constraint Language*, an RDF vocabulary for describing RDF graph structures. Some of these graph structures are captured as *shapes*, which group together constraints about the same RDF nodes. Shapes provide a high-level vocabulary to identify predicates and their associated cardinalities, datatypes and other constraints. Additional constraints can be associated with shapes using SPARQL and similar executable languages. These executable languages can also be used to define new high-level vocabulary terms. SHACL shapes can be used to communicate data structures associated with some process or interface, generate or validate data, or drive user interfaces.

5. Conclusion and Future Work

In this paper, we outlined our idea of a general framework to support the mapping of high-level constraint languages to a generic representation, which can directly be validated by providing a mapping from the generic representation to SPIN/SPARQL queries to actually validate data against constraints provided in the high-level language. The framework consists of a very simple conceptual model using the *RDF Constraints Vocabulary (RDF-CV)* which has been introduced in this paper. The core of the framework is the definition of 103 constraining elements that are used to define constraints of all 81 constraint types that to date have been identified within the DCMI RDF Application Profiles Task Group and in cooperation with the W3C Data Shapes

[19] http://bibframe.org
[20] http://www.xml.com/pub/a/2001/02/07/schemarama.html

https://doi.org/10.23106/dcmi.952137055

Working Group. The full definition of all constraint types and the generic representation of the types in RDF-CV is provided in an accompanying technical report (Bosch et al., 2015).

We have demonstrated how the framework can be used to map a constraint language to RDF-CV and also how to map back from RDF-CV to the constraint language. The latter enables the transformation of semantically equivalent constraints from one constraint language to another via the RDF-CV intermediate representation.

We think that this approach is suitable

1. to implement the validation of constraints consistently across constraint languages,
2. to support the extension of constraint languages when additional constraint types should be supported by means of a simple mapping, and
3. to enhance or rather establish the interoperability of different constraint languages.

It is part of future work to finalize the implementation of all 81 constraint types in our *RDF Validator*, to fully map constraint languages to RDF-CV (first and foremost DSP and OWL 2) and of course keep the framework in sync with the ongoing work in the working groups.

References

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., & Patel-Schneider, P. F. (Eds.). (2003). The Description Logic Handbook: Theory, Implementation, and Applications. New York, NY, USA: Cambridge University Press.

Baader, F., & Nutt, W. (2003). The Description Logic Handbook. In F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, & P. F. Patel-Schneider (Eds.), Basic Description Logics (pp. 43–95). New York, NY, USA: Cambridge University Press. Retrieved from http://dl.acm.org/citation.cfm?id=885746.885749

Boneva, I., Gayo, J. E. L., Hym, S., Prud'hommeau, E. G., Solbrig, H. R., & Staworko, S. (2014). Validating RDF with Shape Expressions. Computing Research Repository (CoRR), abs/1404.1270. Retrieved from http://arxiv.org/abs/1404.1270

Boneva, I., & Prud'hommeaux, E. (2015, July). Core SHACL Semantics (W3C Editor's Draft). W3C. (http://w3c.github.io/data-shapes/semantics/) Bosch, T., & Eckert, K. (2014a). Requirements on RDF Constraint Formulation and Validation. In Proceedings of the DCMI International Conference on Dublin Core and Metadata Applications (DC 2014). Austin, Texas, USA. (http://dcevents.dublincore.org/IntConf/dc-2014/paper/view/257)

Bosch, T., & Eckert, K. (2014b). Towards Description Set Profiles for RDF using SPARQL as Intermediate Language. In Proceedings of the DCMI International Conference on Dublin Core and Metadata Applications (DC 2014). Austin, Texas, USA. (http://dcevents.dublincore.org/IntConf/dc-2014/paper/view/270)

Bosch, T., Nolle, A., Acar, E., & Eckert, K. (2015). RDF Validation Requirements - Evaluation and Logical Underpinning. Computing Research Repository (CoRR), abs/1501.03933. (http://arxiv.org/abs/1501.03933)

Coyle, K., & Baker, T. (2009, May). Guidelines for Dublin Core Application Profiles (DCMI Recommended Resource). Dublin Core Metadata Initiative (DCMI). Retrieved from http://dublincore.org/documents/2009/05/18/profile-guidelines/ (http://dublincore.org/documents/2009/05/18/profile-guidelines/)

Fürber, C., & Hepp, M. (2010). Using SPARQL and SPIN for Data Quality Management on the Semantic Web. In W. Abramowicz & R. Tolksdorf (Eds.), Business Information Systems (Vol. 47, pp. 35–46). Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-642-12814-1 4 doi: 10.1007/978-3-642-12814-1 4

Godby, Carol Jean and Denenberg, Ray. (2015, January). Common Ground: Exploring Compatibilities Between the Linked Data Models of the Library of Congress and OCLC (Tech. Rep.). Library of Congress. Retrieved from http://www.oclc.org/research/publications/2015/oclcresearch-loc-linked-data-2015.html (http://www.oclc.org/research/publications/2015/oclcresearch-loc-linked-data-2015.html)

Harris, S., & Seaborne, A. (2013, March). SPARQL 1.1 Query Language (W3C Recommendation). W3C. Retrieved from http://www.w3.org/TR/2013/REC-sparql11-query -20130321/ (http://www.w3.org/TR/2013/REC-sparql11-query-20130321/)

Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P. F., & Rudolph, S. (2012, December). OWL 2 Web Ontology Language Primer (Second Edition) (W3C Recommendation). W3C. Retrieved from http://www.w3.org/TR/2012/REC-owl2-primer-20121211/ (http://www.w3.org/TR/2012/REC-owl2-primer-20121211/)

https://doi.org/10.23106/dcmi.952137055

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., & Dean, M. (2004). SWRL: A Semantic Web Rule Language Combining OWL and RuleML (W3C Member Submission). W3C. W3C Member Submission. Retrieved from http://www.w3.org/Submission/SWRL (http://www.w3.org/Submission/SWRL)

ISO/IEC. (2006, June). ISO/IEC 19757-3:2006 - Information Technology — Document Schema Definition Languages (DSDL) - Part 3: Rule-Based Validation - Schematron (ISO/IEC Specification). ISO/IEC. (http://standards.iso.org/ittf/PubliclyAvailableStandards/c040833 ISO IEC 19757-3 2006(E).zip)

Knublauch, H. (2015, July). Shapes Constraint Language (SHACL) (W3C Editor's Draft). W3C. (http://w3c.github.io/data-shapes/shacl/) Knublauch, H., Hendler, J. A., & Idehen, K. (2011, February). SPIN - Overview and Motivation (W3C Member Submission). W3C. (http://www.w3.org/Submission/2011/SUBM-spin-overview-20110222/)

Kontokostas, D., Westphal, P., Auer, S., Hellmann, S., Lehmann, J., Cornelissen, R., & Zaveri, A. (2014). Test-driven Evaluation of Linked Data Quality. In Proceedings of the 23rd International World Wide Web Conference (WWW 2014) (pp. 747–758). Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee. Retrieved from http://dx.doi.org/10.1145/2566486.2568002 doi: 10.1145/2566486.2568002

Kroeger, A. (2013). The Road to BIBFRAME: The Evolution of the Idea of Bibliographic Transition into a Post-MARC Future. Cataloging & Classification Quarterly, 51(8), 873-890. Retrieved from http://dx.doi.org/10.1080/01639374.2013.823584 doi:10.1080/01639374.2013.823584

Krötzsch, M., Simanˇ c´ık, F., & Horrocks, I. (2012). A Description Logic Primer. In J. Lehmann & J. Völker (Eds.), Perspectives on Ontology Learning. IOS Press. Library of Congress. (2014a, April). BIBFRAME Authorities (Library of Congress Draft Specification). Library of Congress. Retrieved from http://www.loc.gov/bibframe/docs/bibframe-authorities.html (http://www.loc.gov/bibframe/docs/bibframe-authorities.html)

Library of Congress. (2014b, May). BIBFRAME Profiles: Introduction and Specification (Library of Congress Draft). Library of Congress. Retrieved from http://www.loc.gov/bibframe/docs/bibframe-profiles.html (http://www.loc.gov/bibframe/docs/bibframe-profiles.html)

Library of Congress. (2014c, April). BIBFRAME Relationships (Library of Congress Draft Specification). Library of Congress. Retrieved from http://www.loc.gov/bibframe/docs/bibframe-relationships.html (http://www.loc.gov/bibframe/docs/bibframe-relationships.html)

Lutz, C., Areces, C., Horrocks, I., & Sattler, U. (2005, June). Keys, Nominals, and Concrete Domains. Journal of Artificial Intelligence Research, 23(1), 667–726. (http://dl.ac.org/citation.cfm?id=1622503.1622518)

Miller, L. (2001, February). RDF Squish Query Language and Java Implementation (Draft). Retrieved from http://ilrt.org/discovery/2001/02/squish/ (http://ilrt.org/discovery/2001/02/squish/)

Miller, Eric and Ogbuji, Uche and Mueller, Victoria and MacDougall, Kathy. (2012, November). Bibliographic Framework as a Web of Data: Linked Data Model and Supporting Services (Tech. Rep.). Washington, DC, USA: Library of Congress. Retrieved from http://www.loc.gov/bibframe/pdf/marcld-report-11-21-2012.pdf (http://www.loc.gov/bibframe/pdf/marcld-report-11-21-2012.pdf)

Nilsson, M. (2008, March). Description Set Profiles: A Constraint Language for Dublin Core Application Profiles (DCMI Working Draft). Dublin Core Metadata Initiative (DCMI). Retrieved from http://dublincore.org/documents/2008/03/31/dc-dsp/(http://dublincore.org/documents/2008/03/31/dc-dsp/)

Nilsson, M., Baker, T., & Johnston, P. (2008, January). The Singapore Framework for Dublin Core Application Profiles (DCMI Recommended Resource). Dublin Core Metadata Initiative (DCMI). Retrieved from http://dublincore.org/documents/2008/01/14/singapore-framework/ (http://dublincore.org/documents/200801/14/singapore-framework/)

Nilsson, M., Powel, A., Johnston, P., & Naeve, A. (2008, January). Expressing Dublin Core Metadata using the Resource Description Framework (RDF) (DCMI Recommendation). Dublin Core Metadata Initiative (DCMI). Retrieved from http://dublincore.org/documents/2008/01/14/dc-rdf/ (http://dublincore.org/documents/2008/01/14/dc-rdf/)

Powell, A., Nilsson, M., Naeve, A., Johnston, P., & Baker, T. (2007, June). DCMI Abstract Model (DCMI Recommendation). Dublin Core Metadata Initiative (DCMI). Retrieved from http://dublincore.org/documents/2007/06/04/abstract-model/ (http://dublincore.org/documents/2007/06/04/abstract-model/)

Prud'hommeaux, E. (2014, June). Shape Expressions 1.0 Primer (W3C Member Sumission). W3C. Retrieved from http://www.w3.org/Submission/2014/SUBM-shex-primer-20140602/ (http://www.w3.org/Submission/2014/SUBM-shex-primer-20140602/)

Prud'hommeaux, E. (2015, July). SHACL-SPARQL (W3C Editor's Draft). W3C. (http://w3c.github.io/data-shapes/semantics/SPARQL)

Prud'hommeaux, E., Labra Gayo, J. E., & Solbrig, H. (2014). Shape Expressions: An RDF Validation and Transformation Language. In Proceedings of the 10th International Conference on Semantic Systems

https://doi.org/10.23106/dcmi.952137055

(SEMANTiCS) (pp. 32–40). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/2660517.2660523 doi: 10.1145/2660517.2660523

Ryman, A. (2014, February). Resource Shape 2.0 (W3C Member Submission). W3C. Retrieved from http://www.w3.org/Submission/2014/SUBM-shapes-20140211/ (http://www.w3.org/Submission/2014/SUBM-shapes-20140211/)

Ryman, A. G., Hors, A. L., & Speicher, S. (2013). OSLC Resource Shape: A Language for Defining Constraints on Linked Data. In C. Bizer, T. Heath, T. Berners-Lee, M. Hausenblas, & S. Auer (Eds.), Proceedings of the International World Wide Web Conference (WWW), Workshop on Linked Data on the Web (LDOW) (Vol. 996). CEUR-WS.org. Retrieved from http://dblp.uni-trier.de/db/conf/www/ldow2013.html#RymanHS13

Schneider, M. (2009, October). OWL 2 Web Ontology Language RDF-Based Semantics (W3C Recommendation). W3C. (http://www.w3.org/TR/2009/REC-owl2-rdf-based-semantics-20091027/)

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., & Katz, Y. (2007). Pellet: A Practical OWL-DL Reasoner. Web Semantics: Science, Services and Agents on the World Wide Web, 5(2), 51–53.

Solbrig, H., & Prud'hommeaux, E. (2014, June). Shape Expressions 1.0 Definition (W3C Member Submission). W3C. Retrieved from http://www.w3.org/Submission/2014/SUBM-shex-defn-20140602/ (http://www.w3.org/Submission/2014/SUBM-shex-defn-20140602/)

https://doi.org/10.23106/dcmi.952137055