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 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 published by today 81 types of constraints that are required by various stakeholders for data applications. We generally use 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 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 demonstrated this approach for Description Set Profiles. In this paper, we generalize from the experience of implementing Description Set Profiles by introducing an abstraction layer that is able to describe any constraint type in a way that is more or less straight-forwardly transformable to SPARQL queries. It provides a basic terminology and classification system for RDF constraints to foster discussions on RDF validation. We demonstrate that using another layer on top of SPARQL helps to implement validation consistently accross constraint languages and simplifies the actual implementation of new 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.*

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)**

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 have been 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* 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)*, *Resource Shapes (ReSh)*, *Shape Expressions (ShEx)*, and the *Web Ontology Language (OWL)*. 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 naming assumptions.

With its direct support of validation via *SPARQL*, the *SPARQL Inferencing Notation (SPIN)* is also very popular to formulate and check constraints (Fürber & Hepp, 2010). We consider SPIN 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 contraint 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 mroe appealing to the data practitioners – producers as well as consumers who again might have different needs and preferences.

We formerly demonstrated that a high-level constraint language, Description Set Profiles, 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.[5]

The creation of such mappings is in many cases not straight-forward and requires profound knowledge of SPARQL, as the following example demonstrates:

[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`

[5]Available at `http://purl.org/net/rdfval-demo`, source code available at: `https://github.com/boschthomas/rdf-validator`.

The constraint type *minimum qualified cardinality restrictions* which corresponds to the requirement *R-75*[6] can be used 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:

```
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:maxOccur "infinity" ;
          dsp:property author ;
          dsp:nonLiteralConstraint [ dsp:valueClass Person ] ] ] .

SPIN: CONSTRUCT { [ a spin:ConstraintViolation ... . ] } WHERE {
          ?this ?p ?o ; a ?C .
          BIND ( qualifiedCardinality( ?this, ?p, ?C ) AS ?c ) .
          BIND( STRDT ( STR ( ?c ), xsd:nonNegativeInteger ) AS ?cardinality ) .
          FILTER ( ?cardinality < ?minimumCardinality ) . }

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

Note that the SPIN representation is NOT a mapping, but a direct expression of the constraint using a SPARQL CONSTRUCT query that creates a spin:ConstraintViolation if the constraint is violated. The mapping for DSP is much more complicated and can be found in the mappings provided by us.[7]

On the other hand, 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.

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

[6] 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`

[7] Online: `https://github.com/boschthomas/rdf-validation/blob/b6a275fb5d71a92ae33d3b6aadd5f447351214b7/SPIN/DSP_SPIN-Mapping.ttl#L4665`

specific 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 contraint 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 contraint is checked. Our framework therefore proides 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[8], 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 transformations. 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:

$$gc = m_1(sc_\alpha)$$
$$sc_\beta = m'_2(gc)$$

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 mapping generally are not invertible. With an intermediate generic 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 and backward – 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 A Vocabulary to Describe RDF Constraints of any RDF Constraint Type

Prerequisites to develop a vocabulary to describe RDF constraints of any RDF constraint type are (1) to specify the underlying semantics for RDF validation and thus for the vocabulary and (2) to define a basic terminology and classification system for RDF constraints which lay the ground for discussions on RDF validation.

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

3.1 Semantics for RDF Validation

RDF validation and OWL 2 assume different semantics. This ambiguity in semantics is one of the main reasons why OWL 2 has not been adopted as a standard constraint language for RDF validation in the past. We compare semantics of RDF validation and OWL 2 as OWL 2 is considered as a constraint language in case the semantics for RDF validation is applied.

OWL 2 is based on the *open-world assumption (OWA)*, i.e., a statement cannot be inferred to be false if it cannot be proved to be true which fits its primary design purpose to represent knowledge on the World Wide Web. As each book must have a title ($\texttt{Book} \sqsubseteq \exists\ \texttt{title}.\top$) and *Hamlet* is a book, *Hamlet* must have a title as well. In an OWA setting, this constraint does not cause a violation, even if there is no explicitly defined title, since there must be a title for this book which we may not know. As RDF validation has its origin in the XML world, many RDF validation scenarios require the *closed-world assumption (CWA)*, i.e., a statement is inferred to be false if it cannot be proved to be true. Thus, classical constraint languages are based on the CWA where constraints need to be satisfied only by named individuals. Since there is no explicitly defined title for the book *Hamlet*, the CWA yields to a violation.

RDF Validation requires that different names represent different objects (*unique name assumption (UNA)*), whereas, OWL 2 is based on the *non-unique name assumption (nUNA)*. If we define the property *title* to be functional (`funct (title)`), a book can have at most one distinct title. UNA causes a clash if the book *Huckleberry-Finn* has more than one title. For nUNA, however, reasoning concludes that the title *The-Adventures-of-Huckleberry-Finn* must be the same as the title *Die-Abenteuer-des-Huckleberry-Finn* which resolves the violation.

If RDF validation would assume OWA and nUNA just like OWL 2 does, validation won't be that restrictive and therefore we won't get the intended validation results. Differences in semantics may lead to differences in validation results when applied to particular constraint types. This is important as for 56.8% of the constraint types validation results differ if the CWA or the OWA is assumed and as for 66.6% of the constraint types validation results are different in case the UNA or the nUNA is assumed (Bosch et al., 2015).

Constraints are identical w.r.t. RDF semantics, 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. As we use multiple languages to express constraints of constraint types, we thereby document identical semantics of these languages with regard to given constraint types (Bosch et al., 2015). This is also important in order to prove that semantically equivalent *specific constraints* and corresponding *generic constraints* behave in the same way w.r.t. validation results which semantically underpins bidirectional mappings.

3.2 Basic Terminology and Classification System for RDF Constraints

A *constraint language* is a language which is used to formulate constraints. The W3C working group defines *RDF constraint* as a component of a schema what needs to be satisfied[9]. As there are already five promising constraint languages, our purpose is not to invent a new language. As there is no high-level declarative language which meets all RDF validation requirements, we rather developed a very simple lightweight vocabulary, consisting of only three classes, three object properties, and three data properties, which is universal

[9] `https://www.w3.org/2014/data-shapes/wiki/Glossary`

enough to describe constraints of any constraint type. We call this vocabulary the *RDF Constraints Vocabulary (RDF-CV)*[10] whose conceptual model is shown in Figure 1.

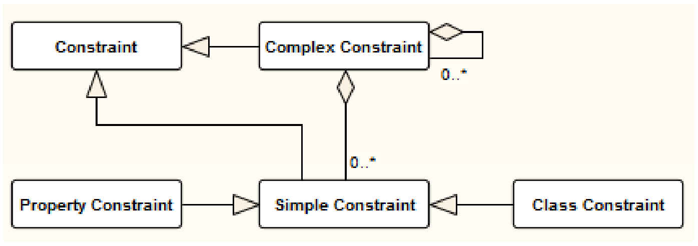

FIG. 1: *RDF Constraints Vocabulary (RDF-CV)* Conceptual Model

We classify sets of constraints according to four dimensions. For each constraint type, we evaluated (Bosch et al., 2015) in which set of constraints their instances are contained.

***Universality*:** Constraints are expressible either generically (*generic constraints*) using the *RDF-CV* or specifically (*specific constraints*) by domain-specific constraint languages. As we express constraints of each constraint type in form of *generic constraints* conforming to the *RDF-CV* (Bosch et al., 2015), we show that constraints of any constraint type can be described generically.

***DL Expressivity*:** The *RDF-CV* allows to describe constraints which are either *expressible in DL* (64%) or which are *not expressible in DL* (36%). Constraints which are expressible in DL are instantiated from 64% of the overall 81 constraint types. On the other hand, constraints which are not expressible in DL are assigned to 36% of all constraint types.

***Complexity*:** *Simple constraints* (60% of the constraint types) denotes the set of atomic constraints. *Complex constraints* (26%) is the set of constraints which are created out of *simple* and/or other *complex constraints*. 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.

***Context*:** *Simple constraints* are applied on properties (*property constraints*, 60%), on classes (*class constraints*, 25%), or on properties and classes (*property and class constraints*, 15%). *Simple constraints* must hold for individuals of their associated *context classes*. As *context classes* of *simple constraints* are reused within *complex constraints*, it does not make sense to have terms standing for *simple* and *complex constraints* within the *RDF-CV*. As a consequence, the distinction of *property* and *class constraints* is sufficient to generically describe constraints of all constraint types.

Altogether, instances of the most of the constraint types are *simple constraints on properties* which are *expressible in DL*.

3.3 Simple Constraints

The implementation model of the *RDF-CV* (Figure 2) shows that both *property constraints* and *class constraints* are *simple constraints*.

[10]Online available at: `https://github.com/boschthomas/RDF-Constraints-Vocabulary`

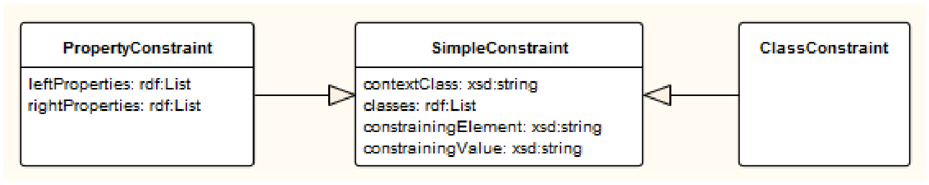

FIG. 2: *RDF Constraints Vocabulary (RDF-CV)* Implementation Model

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

TABLE 2: Minimum Qualified Cardinality Restriction as Property Constraint

constraint set	context class	left property list	right p. list	classes	constraining element	c. value
property	Publication	author	-	Person	$\geq$	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 the constraint type *property domain (R-25, R-26)* whose constraints restrict domains of properties, e.g., there is only one constraining element with exactly the same identifier *property domain*. For a few constraint types, however, one constraining element is not sufficient to describe all possible constraints of a particular constraint type and therefore multiple constraining elements may be stated. The constraint type *language tag cardinality (R-48, R-49)*, for instance, 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.

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 (e.g., *minInclusive*) as constraining elements. For some constraint types like *minimum qualified cardinality restrictions (R-75)*, it is more intuitive and concise to directly apply DL constructs like the *at-least restriction* ($\geq$) as constraining elements. We provide a complete list of all constraining elements which can be used to express constraints of any constraint type (Bosch et al., 2015).

In some cases, a constraint is only complete when a *constraining value* is stated in addition to the constraining element and simple constraints may refer to a list of *classes*. The constraining element of the property constraint `Publication` $\sqsubseteq$ $\geq$`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.

For property constraints, *left* and *right property lists* are specified. The assignment of properties to these lists happens relative to 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` $\circ$ `genre` $\sqsubseteq$ `authorOfGenre` infers that *Stephen-Hawking* is an author of the genre *Popular-Science*. Thus, when representing the property constraint using the RDF-CV (see Table 3), the properties *authorOf* and *genre* are placed on the left side of the constraining element *property paths* and the property *authorOfGenre* on its right side. As this property 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

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

There are simple constraints which 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 2 can be used to formulate the constraint:

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

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.

Table 4 presents (1) in the first line how the literal pattern matching simple constraint which is not expressible in DL and (2) in the second line how the universal quantification complex constraint which is expressible in DL are represented using the RDF-CV. Thereby, the context class *ISBN*, whose instances must satisfy the simple constraint, is reused within the list of classes the complex constraint 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

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

3.4 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 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*:

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

As *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:

$$\begin{aligned} \textbf{Publication} \sqsubseteq (\neg\mathbf{E} \sqcap \mathbf{F}) \sqcup (\mathbf{E} \sqcap \neg\mathbf{F}) \text{ , E} \equiv \text{A} \sqcap \text{B , F} \equiv \text{C} \sqcap \text{D} \\ \text{A} \sqsubseteq \geq 1 \text{ isbn.string} \sqcap \leq 1 \text{ isbn.string , B} \sqsubseteq \geq 1 \text{ title.string} \sqcap \leq 1 \text{ title.string} \\ \text{C} \sqsubseteq \geq 1 \text{ issn.string} \sqcap \leq 1 \text{ issn.string , D} \sqsubseteq \geq 1 \text{ title.string} \sqcap \leq 1 \text{ title.string} \end{aligned}$$

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 (=n)* and *exclusive or (XOR) of property groups* are 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 (R-13)* 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

c. set	context class	left p. list	right p. list	classes	c. element	c. value
class	Publication	-	-	E, F	XOR of property groups	-
class	E	-	-	A, B	intersection	-
class	F	-	-	C, D	intersection	-
property	A	isbn	-	string	=	1
property	B	title	-	string	=	1
property	C	issn	-	string	=	1
property	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).

TABLE 6: Primary Key Properties as Complex Constraints

c. set	context class	left p. list	right p. list	classes	c. element	c. value
property	$\top$	isbn^{-}	isbn^{-}	-	inverse property	-
property	Book	isbn^{-}	-	-	functional property	-

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

```
[   a rdfcv:PropertyConstraint , rdfcv:SimpleConstraint ;
    rdfcv:contextClass Book ; rdfcv:leftProperties ( isbn ) ; rdfcv:constrainingElement "keyfor" ] .
```

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 (Bosch & Eckert, 2015).

Constraints of the *irreflexive object properties (R-60)* constraint type ensure that no individual is connected by a given object property to itself (Krötzsch, Simančík, & Horrocks, 2012). With the irreflexive object property constraint $\top \sqsubseteq \neg \exists authorOf.Self$, 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 - one property and two class constraints (see Table 7).

TABLE 7: Irreflexive Object Properties as Complex Constraints

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

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

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

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 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)*[11] (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).

Stardog Integrity Constraint Validation (ICV) and the *Pellet Integrity Constraint Val-*

[11]http://spinrdf.org

idator (ICV) use OWL 2 constructs to formulate constraints. The Pellet ICV[12] is a proof-of-concept extension for the OWL 2 DL reasoner *Pellet* (Sirin, Parsia, Grau, Kalyanpur, & Katz, 2007). Stardog ICV[13] 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* 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 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*[14] (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[15] 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 (International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), 2006), an XML schema and XML structure validation language which

[12]`http://clarkparsia.com/pellet/icv`
[13]`http://docs.stardog.com/#_validating_constraints`
[14]`http://bibframe.org`
[15]`http://www.xml.com/pub/a/2001/02/07/schemarama.html`

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

Based on our work in the *DCMI* and in cooperation with the *W3C* working group, we published by today 81 requirements to validate RDF data and to formulate constraints; each of them corresponds to a constraint type from which concrete constraints are instantiated to be checked on RDF data. These constraint types form the basis to lay the ground for discussions on validation by defining a basic terminology and classification system for RDF constraints. There exists no single best solution considered as high-level intuitive constraint language which enables to express constraints in an easy and concise way and which is able to meet all requirements raised by data practitioners. Thus, the idea behind this paper is to satisfy all requirements by representing constraints of any constraint type in a generic way using a lightweight vocabulary which consists of only a few terms.

As there is no standard way to formulate constraints, semantically equivalent *specific constraints* may be represented by a variety of languages - each of them having different syntax and semantics. We propose transformations between semantically equivalent *specific constraints* by using the proposed vocabulary to intermediately represent constraints in a generic way (1) to avoid the necessity to understand several languages, (2) to resolve misunderstandings about the meaning of particular constraints, and (3) to enhance the interoperability of constraint languages (Section **??**).

We use *SPIN* as basis to develop a validation environment[??] to validate RDF data according to constraints of constraint types which are expressible by arbitrary constraint languages by mapping them to *SPIN*[??] (Bosch & Eckert, 2014b). When language designers extend constraint languages to be able to formulate constraints of not yet supported constraint types, our proposal enables to reuse the validation implementation of these constraint types by mapping *specific constraints* expressed by the language to the corresponding *generic constraint*. For any constraint language, we enable to offer a validation implementation of any constraint type out-of-the-box by providing a *SPIN* mapping for each constraint type[8] whose constraints are represented generically (Bosch & Eckert, 2015). Furthermore, our proposal makes sure that the validation on semantically equivalent *specific constraints* expressed by different languages leads to exactly the same validation results, i.e., that validation is performed independently from the used language (Section **??**).

It is part of future work (1) to extend the *RDF Validator* by generating *generic constraints* according to inputs of domain experts who may not be familiar with the formulation of constraints, (2) to offer bidirectional transformations between *specific constraints* expressed by the most common constraint languages and corresponding *generic constraints*, and (3) to provide translations between semantically equivalent *specific constraints* expressed by the most common constraint languages by using the developed vocabulary to intermediately represent constraints. Identifying RDF validation requirements is an ongoing process, so the task to map constraints of new constraint types to the vocabulary.

References

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. Retrieved from `http://dcevents.dublincore.org/IntConf/dc-2014/paper/view/257` (`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.

Bosch, T., & Eckert, K. (2015). Expressing RDF Constraints Generically. In *Proceedings of the DCMI International Conference on Dublin Core and Metadata Applications (DC 2015).* São Paulo, Brazil.

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

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)

International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC). (2006, June). *ISO/IEC 19757-3:2006 - Information Technology — Document Schema Definition Languages (DSDL) - Part 3: Rule-Based Validation - Schematron* (International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) Specification). International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC). Retrieved from http://standards.iso.org/ittf/PubliclyAvailableStandards/c040833_ISO_IEC_19757-3_2006(E).zip (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., Simančík, 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. Retrieved from http://dl.acm.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/2008/01/14/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 Submission). 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 (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. Retrieved from `http://www.w3.org/TR/2009/REC-owl2-rdf-based-semantics-20091027/` (`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/`)