International Journal of Semantic Computing

DIRECTING THE DEVELOPMENT OF CONSTRAINT LANGUAGES BY CHECKING CONSTRAINTS ON RDF DATA

THOMAS HARTMANN

Monitoring Society and Social Change, GESIS - Leibniz Institute for the Social Sciences, Square B2 1, Mannheim, 68159, Germany
thomas.hartmann@gesis.org
http://www.gesis.org

BENJAMIN ZAPILKO

Knowledge Technologies for the Social Sciences, GESIS - Leibniz Institute for the Social Sciences, Unter Sachsenhausen 6-8
Cologne, 50667, Germany
benjamin.zapilko@gesis.org

JOACHIM WACKEROW

Monitoring Society and Social Change, GESIS - Leibniz Institute for the Social Sciences, Square B2 1, Mannheim, 68159, Germany
joachim.wackerow@gesis.org

KAI ECKERT

WISS Research Group, Faculty of Information and Communication, Stuttgart Media University, Nobelstraße 10, Stuttgart, 70569, Germany
eckert@hdm-stuttgart.de

Received (15/03/2016)
Revised (Day Month Year)
Accepted (Day Month Year)

For research institutes, data libraries, and data archives, validating RDF data according to predefined constraints is a much sought-after feature, particularly as this is taken for granted in the XML world. Based on our work in two international working groups on RDF validation and jointly identified requirements to formulate constraints and validate RDF data, we have published 81 types of constraints that are required by various stakeholders for data applications.

In this paper, we evaluate the usability of identified constraint types for assessing RDF data quality by (1) collecting and classifying 115 constraints on vocabularies commonly used in the social, behavioral, and economic sciences, either from the vocabularies themselves or from domain experts, and (2) validating 15,694 data sets (4.26 billion triples) of research data against these constraints. We classify each constraint according to (1) the severity of occurring violations and (2) based on which types of constraint languages are able to express its constraint type. Based on the large-scale evaluation, we formulate several findings to direct the further development of constraint languages.

Keywords: RDF Data Validation; RDF Data Quality; Constraint Languages; Semantic Web; Linked Data; RDF.

1. Introduction

For constraint formulation and RDF data validation, several languages exist or are currently developed. *Shape Expressions (ShEx)*, *Resource Shapes (ReSh)*, *Description Set Profiles (DSP)*, the *Web Ontology Language (OWL)*, the *SPARQL Inferencing Notation (SPIN)*, and the *SPARQL Query Language for RDF* are the six most promising and widely used constraint languages. OWL is used as a constraint language under the closed-world and unique name assumptions. With its direct support of validation via SPARQL, SPIN is very popular and certainly plays an important role for future developments in this field. It is particularly interesting as a means to validate arbitrary constraint languages by mapping them to SPARQL [1]. In addition, the W3C currently develops the *Shapes Constraint Language (SHACL)*, an RDF vocabulary for describing RDF graph structures. Yet, there is no clear favorite and none of the languages is able to meet all requirements raised by data practitioners. This is the reason why further research on RDF validation and the development of constraint languages is needed.

In 2013, the W3C organized the RDF Validation Workshop, where experts from industry, government, and academia discussed first use cases for constraint formulation and RDF data validation. In 2014, two working groups on RDF validation have been established to develop a language to express constraints on RDF data: the *W3C RDF Data Shapes Working Group* and the *DCMI RDF Application Profiles Task Group* which among others bundles the requirements of data institutions of the cultural heritage sector and the *social, behavioral, and economic (SBE)* sciences and represents them in the W3C working group.

Within the DCMI working group, a collaboratively curated database of RDF validation requirements, online available at http://purl.org/net/rdf-validation, has been created which contains the findings of the working groups based on various case studies provided by various data institutions [2]. The database connects requirements to use cases, case studies, and solutions and forms the basis of this paper. Based on our work in these working groups and jointly identified requirements to formulate constraints and validate RDF data, we have published 81 types of constraints; each of them corresponding to a requirement in the database.

In this paper, we collected 115 constraints for three vocabularies commonly used in the SBE domain (see Sec. 2), either from the vocabularies themselves or from several domain and data experts, in order to gain a better understanding about the role of certain constraint types for assessing data quality. We let the experts classify the constraints according to the severity of their violation. Furthermore, we classified each constraint type based on whether it is expressible by RDFS/OWL, common high-level constraint languages, or only by plain SPARQL (see Sec. 4).

As we do not want to base our conclusions on the evaluation of vocabularies and constraint definitions alone, we conducted a large-scale experiment. For all these 115 constraints on the vocabularies DDI-RDF, QB, and SKOS, we evaluated the data quality of 15,694 data sets (4.26 billion triples) of SBE research data, obtained from 33 SPARQL endpoints. Based on the evaluation results, we formulate several findings to direct the further development of constraint languages. To make valid general statements for all vocabularies, however, these findings still have to be verified or falsified by evaluating the quality of data represented by more than three vocabularies (see Sec. 6).

In Sec. 5, we delineate how we implemented our validation environment which can directly be used to validate RDF data against constraints expressed in any RDF-based constraint language and extracted from or defined for any RDF vocabulary.

In this paper, we discuss constraints on RDF data in general. Note that the data represented in RDF can be data in the sense of the SBE sciences, but also metadata about published or unpublished data. We generally refer to both simply as RDF data and only distinguish between data and metadata in the data set descriptions.

2. Common Vocabularies in the SBE Sciences

We took all well-established and newly developed SBE vocabularies into account and defined constraints for three vocabularies commonly used in the SBE sciences. We analyzed actual data according to constraint violations, as for these vocabularies large data sets have already been published.

The data most often used in research within SBE sciences is *unit-record data*, i.e., data collected about individuals, businesses, and households, in form of responses to studies or taken from administrative registers such as hospital records, registers of births and deaths. A *study* represents the process by which a data set was generated or collected. The range of unit-record data is very broad - including census, education, and health data and business, social, and labor force surveys. This type of research data is held within data archives or data libraries after it has been collected, so that it may be reused by future researchers.

2.1. *Vocabularies for representing multi-dimensional aggregated data and its metadata*

By its nature, unit-record data is highly confidential and access is often only permitted for qualified researchers who must apply for access. Researchers typically represent their results as aggregated data in form of multi-dimensional tables with only a few columns, so-called *variables* such as *sex* or *age*. Aggregated data, which answers particular research questions, is derived from unit-record data by statistics on groups or aggregates such as frequencies and arithmetic means. The purpose of publicly available aggregated data is to get a first overview and to gain an interest in further analyses on the underlying unit-record data. Aggregated data is published in form of CSV files, allowing to perform calculations on the data.

For more detailed analyses, researchers refer to unit-record data including additional variables needed to answer subsequent research questions like the comparison of studies between countries. *Eurostat*, the statistical office of the European Union, provides research findings in form of aggregated data (downloadable as CSV files) and its metadata on European level that enable comparisons between countries. *Formal childcare* is an example of an aggregated variable which captures the measured availability of childcare services in percent over the population in European Union member states by the dimensions *year*, *duration*, *age* of the child, and *country*. Variables are constructed out of values (of one or multiple datatypes) and/or code lists. The variable *age*, e.g., may be represented by values of the datatype *xsd:nonNegativeInteger* or by a code list of age clusters (e.g., '0 to 10' and '11 to 20').

A representative RDF validation case study within the SBE sciences is to ensure correctness when comparing variables between data collections of different countries. Several vocabulary-specific constraints on RDF data are checked for each data collection in order to determine if variables measuring *age* - collected for different countries (age_{DE}, age_{UK}) - are comparable: (1) variable definitions must be available, (2) for each code a human-readable label has to be specified, (3) code lists must be structured properly, and (4) code lists must either be identical or at least similar. If a researcher only wants to get a first overview on the comparability of variables (use case 1), covering the first three constraints may be sufficient, i.e., the violation of the first three constraints is more serious than the violation of the last constraint. If the intention of the researcher is to perform more sophisticated comparisons (use case 2), however, the user may raise the severity level of the last constraint.

The *RDF Data Cube Vocabulary (QB)* [3] is a W3C recommendation for representing *data cubes*, i.e., multi-dimensional aggregated data and its metadata, in RDF [4]. A *qb:DataStructureDefinition* contains metadata on the data collection. The variable *formal childcare* is modeled as *qb:measure*, since it stands for what has been measured in the data collection. *Year*, *duration*, *age*, and *country* are *qb:dimensions*. Data values, i.e., the availability of childcare services in percent over the population, are collected in a *qb:DataSet*. Each data value is represented inside a *qb:Observation* containing the values for each dimension [5].

The development of QB is based on the *Statistical Core Vocabulary (SCOVO)* [6, 7, 4], an RDFS-based, lightweight, and extensible vocabulary for representing statistical data on the Web. SCOVO offers a basic core of classes and properties for representing data sets, multiple dimensions, and statistical items. [8] extend SCOVO with a vocabulary enabling the connection of SCOVO-described data sets with external vocabularies to perform more complex data analyses and improve discoverability and reusability.

2.2. *Vocabulary for representing metadata on data in tabular format*

Physical Data Description (PHDD) [9] is a vocabulary to represent metadata about data in tabular format as well as its physical properties in RDF, enabling further aggregations and calculations. The data could either be represented in records with character-separated values (CSV) or fixed length. PHDD is usable standalone or together with related vocabularies like DDI-RDF or DCAT. The combined usage of PHDD, DDI-RDF, and DCAT enables the creation of data repositories providing metadata for the description of collections, data discovery, and the processing of the data. Descriptions in PHDD could be added to web pages which enables an automatic processing of the data by programs.

Eurostat provides a CSV file, a two-dimensional table (*phdd:Table*) about the variable *formal childcare* which is structured by a table structure (*phdd:TableStructure*, *phdd:Delimited*) including information about the character set (*ASCII*), the variable delimiter (,), the new line marker (*CRLF*), and the first line where the data starts (*2*). The table structure is related to table columns (*phdd:Column*) which are described by column descriptions (*phdd:DelimitedColumnDescription*). For the column containing the cell values in percent, the column position (*5*), the recommended data type (*xsd:nonNegativeInteger*), and the storage format (*TINYINT*) are given.

2.3. *Vocabulary for representing metadata on unit-record data*

The SBE sciences require high-quality data for their empirical research. For more than a decade, members of the SBE community have been developing and using a metadata standard, composed of almost twelve hundred metadata fields, known as the *Data Documentation Initiative (DDI)* [10], an XML format to disseminate, manage, and reuse data collected and archived for research. In XML, the definition of schemas containing constraints and the validation of data according to these constraints is commonly used to ensure a certain level of data quality. With the rise of the Web of Data, data professionals and institutions are very interested in publishing their data directly in RDF or at least publish accurate metadata about their data to facilitate discovery and reuse. Recently, members of the SBE and Linked Data community developed with the *DDI-RDF Discovery Vocabulary (DDI-RDF)* [11] a means to expose DDI metadata as Linked Data.

For more detailed analyses, we refer to the unit-record data collected for the series *EU-SILC (European Union Statistics on Income and Living Conditions)*. Where data collection is cyclic, data sets may be released as *series*, where each cycle produces one or more data sets. The aggregated variable *formal childcare* is calculated on the basis of six unit-record variables (i.a., *Education at pre-school*) for which detailed metadata is given (i.a., code lists) enabling researchers to replicate the results shown in aggregated data tables.

DDI-RDF is used to represent metadata on unit-record data in RDF. The se-

ries (*disco:StudyGroup*) EU-SILC contains one study (*disco:Study*) for each year (*dcterms:temporal*) of data collection. The property *dcterms:spatial* points to the countries for which the data has been collected. The study *EU-SILC 2011* contains eight unit-record data sets (*disco:LogicalDataSet*) including unit-record variables (*disco:Variable*) like the six ones needed to calculate the aggregated variable *formal childcare*.

2.4. *Vocabularies for representing knowledge organization systems and formal statistical classifications*

The *Simple Knowledge Organization System (SKOS)* [12, 13] is a vocabulary to represent knowledge organization systems such as thesauri, classification schemes, and taxonomies. Using SKOS, a knowledge organization system is expressible as machine-readable data in a machine-processable standard format for the Semantic Web. It can then be exchanged between computer applications and published in a machine-readable format in the Web [13]. SKOS provides high interoperability with other standards, formats, and applications as it is based on RDF, the standard model for data interchange on the Web. SKOS is used to represent term relations, term hierarchies, and the structure and semantics of vocabularies. A vocabulary is typically represented as a *skos:ConceptScheme* that holds multiple *skos:Concepts* which can be linked to other *skos:Concepts* by hierarchical and associative properties that are oriented on relations of the ISO norms for thesauri like *skos:broader*, *skos:narrower*, and *skos:related*.

Thesauri organize complex relations between terms even on the lexical level. The *SKOS Simple Knowledge Organization System eXtension for Labels (SKOS-XL)* [14] defines an extension for SKOS, providing additional support for describing and linking lexical entities. This provides a complexity of relationships between terms which is needed by several vocabularies.

SKOS is reused multiple times to build SBE vocabularies. The codes of the variable *Education at pre-school* (measuring the number of education hours per week) are modeled as *skos:Concepts* and a *skos:OrderedCollection* organizes them in a particular order within a *skos:memberList*. A variable may be associated with a theoretical concept (*skos:Concept*). Hierarchies of theoretical concepts are built within a *skos:ConceptScheme* of a series using *skos:narrower*. The variable *Education at pre-school* is assigned to the theoretical concept *Child Care* which is the narrower concept of *Education*, one of the top concepts of the series EU-SILC. Controlled vocabularies (*skos:ConceptScheme*), serving as extension and reuse mechanism, organize types (*skos:Concept*) of descriptive statistics (*disco:SummaryStatistics*) like minimum, maximum, and arithmetic mean.

The *SKOS Extension for Statistics (XKOS)* [15] is a vocabulary to describe formal statistical classifications and introduce refinements of SKOS semantic properties [16]. A formal statistical classification is a hierarchical concept scheme including concepts, associated codes (numeric string labels), short textual labels, definitions,

and longer descriptions that include rules for their use.

XKOS extends SKOS for the needs of statistical classifications like the *International Standard Classification of Occupations (ISCO)* and the *Statistical Classification of Economic Activities in the European Community (NACE)*. It does so in two main directions. First, it defines a number of terms that enable the representation of statistical classifications with their structure and textual properties, as well as different types of relations between classifications and contained concepts. Second, it refines SKOS semantic properties to allow the use of more specific relations between concepts. Those specific relations can be used for the representation of classifications or for any other case where SKOS is employed [17]. The National Institute of Statistics and Economic Studies (Insee) already provides diverse statistical classifications in RDF conforming to XKOS.

3. Related Work

For data archives, research institutes, and data libraries, RDF validation according to predefined constraints is a much sought-after feature, particularly as this is taken for granted in the XML world. DDI-XML documents, e.g., are validated against diverse XML Schemas. As certain constraints cannot be formulated and validated by XML Schemas, so-called secondary-level validation tools like *Schematron* have been introduced to overcome the limitations of XML validation. Schematron generates validation rules and validates XML documents according to them. With RDF validation, one can overcome the drawbacks when validating XML documents. It cannot be validated, e.g., if each code of a variable's code list is associated with a category and that if an element has a specific value then certain child elements must be present.

A well-formed *RDF Data Cube* is an RDF graph describing one or more instances of *qb:DataSet* for which each of the 22 integrity constraints, defined within the QB specification, passes [3]. Each integrity constraint is expressed as narrative prose and, where possible, as a SPARQL ASK query or query template. If the ASK query is applied to an RDF graph then it will return true if that graph contains one or more QB instances which violate the corresponding constraint.

[18] investigated how to support taxonomists in improving SKOS vocabularies by pointing out quality issues that go beyond the integrity constraints defined in the SKOS specification.

Stardog ICV and *Pellet ICV* use OWL 2 constructs to formulate constraints. *OWL* in its current version 2 is an expressive language which is based on formal logic and on the subject-predicate-object triples from RDF. OWL offers knowledge representation and reasoning services in combination with *SWRL*. Validation, however, is not the primary purpose of its design which has lead to claims that OWL cannot be used for validation. [19] and [20], e.g., discuss the differences between constraints and RDFS/OWL axioms. 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 semantics of RDFS/OWL ontologies which could therefore not only be used for reasoning but also for validation.

The semantics which is applied for RDF validation is CWA/UNA. RDF validation requires that different names represent different objects (*unique name assumption (UNA)*), whereas OWL is based on the *non-unique name assumption (nUNA)*. Reasoning in OWL 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. On the other hand, 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. This ambiguity in semantics is one of the main reasons why OWL has not been adopted as a standard constraint language for RDF validation in the past. [21] propose an alternative semantics for OWL using CWA/UNA so that it could be used to validate integrity constraints. [22] claims that DL and therefore OWL axioms can be interpreted in a closed-world setting and used for constraint checking. When using OWL axioms in terms of constraints, we adopt the same semantics that is used for RDF validation.

4. Classification of Constraint Types and Constraints

To gain insights into the role that certain types of constraints play for assessing the quality of RDF data, we use two simple classifications. On the one hand, we classify constraint types based on whether they are expressible by different types of constraint languages. On the other hand, we let several domain experts classify constraints, which are formulated for a given vocabulary, according to the perceived severity of their violation.

For the three vocabularies, several SBE domain experts determined the severity level for each of the 115 constraints. In a technical report, we provide detailed textual descriptions for all these constraints as well as their assignments to constraint types and severity levels [23]. In the following, we summarize the classifications of constraint types and constraints for the purpose of our evaluation.

4.1. *Classification of constraint types according to the expressivity of constraint language types*

According to the expressivity of three different types of constraint languages, the complete set of constraint types encompasses the subsequent three not disjoint sets of constraint types:

(1) *RDFS/OWL Based*
(2) *Constraint Language Based*
(3) *SPARQL Based*

RDFS/OWL Based. *RDFS/OWL Based* denotes the set of constraint types which can be formulated with RDFS/OWL axioms when using them in terms of constraints with CWA/UNA semantics and without reasoning. The entailment regime

is to be decided by the implementers. It is our point that reasoning affects validation and that a proper definition of the reasoning to be applied is needed.

The modeling languages RDFS and OWL are typically used to formally specify vocabularies and RDFS/OWL axioms are commonly found within formal specifications of vocabularies. In general, constraints instantiated from *RDFS/OWL Based* constraint types can be seen as a basic level of constraints ensuring that the data is consistent with the formally and explicitly specified intended semantics of vocabularies as well as with the integrity of vocabularies' conceptual models about data.

Constraints of the type *minimum qualified cardinality restrictions* (*R-75*), e.g., guarantee that individuals of given classes are connected by particular properties to at least n different individuals/literals of certain classes or data ranges. For *DDI-RDF*, a *minimum qualified cardinality restriction* can be obtained from a respective OWL axiom to ensure that each *disco:Questionnaire* includes (*disco:question*) at least one *disco:Question*:

```
OWL:
disco:Questionnaire rdfs:subClassOf
  [ a owl:Restriction ;
    owl:minQualifiedCardinality 1 ;
    owl:onProperty disco:question ;
    owl:onClass disco:Question ] .
```

In contrast to *RDFS/OWL Based* constraints, *Constraint Language* and *SPARQL Based* constraints are usually not (yet) explicitly defined within formal specifications of vocabularies. Instead, they are often specified within formal as well as informal textual descriptions of vocabularies. Additionally, we let domain experts define constraints when they agreed that violating these constraints would affect the usefulness of the data.

Constraint Language Based. We further distinguish *Constraint Language Based* as the set of constraint types that can be expressed by classical high-level constraint languages like ShEx, ReSh, and DSP. There is a strong overlap between *RDFS/OWL* and *Constraint Language Based* constraint types as in many cases constraint types are expressible by both classical constraint languages and RDFS/OWL. SPARQL, however, is considered as a low-level implementation language in this context. In contrast to SPARQL, high-level constraint languages are comparatively easy to understand and constraints can be formulated in a more concise way. Declarative languages may be placed on top of SPARQL when using it as an execution language. For *Constraint Language Based* constraint types, we expect a straight-forward support in future constraint languages.

Context-specific exclusive or of property groups (*R-13*) is a constraint type which can be formulated by the high-level constraint language ShEx. Constraints of this type restrict individuals of given classes to have property links of properties defined within exactly one of multiple mutually exclusive property groups. Within the context of DDI-RDF, e.g., *skos:Concept*s can have either *skos:definition* (when

interpreted as theoretical concepts) or *skos:notation* and *skos:prefLabel* properties (when interpreted as codes), but not both:

```
ShEx:
skos:Concept {
    ( skos:definition xsd:string ) |
    ( skos:notation xsd:string , skos:prefLabel xsd:string ) }
```

SPARQL Based. The set *SPARQL Based* encompasses constraint types that are not expressible by RDFS/OWL or common high-level constraint languages but by plain SPARQL. For assessing the quality of thesauri, e.g., we concentrate on their graph-based structure and apply graph- and network-analysis techniques. An example of such constraints of the constraint type *structure* is that a thesaurus should not contain many orphan concepts, i.e., concepts without any associative or hierarchical relations, lacking context information valuable for search. As the complexity of this constraint is relatively high, it is only expressible by SPARQL, not that intuitive, and quite complex:

```
SPARQL:
SELECT ?concept WHERE {
    ?concept a [ rdfs:subClassOf* skos:Concept ] .
    FILTER NOT EXISTS { ?concept ?p ?o .
        FILTER ( ?p IN ( skos:related, skos:relatedMatch,
                         skos:broader, ... ) ) . } }
```

SPARQL Based constraint types are today only expressible by plain SPARQL. Depending on their usefulness, a support in high-level constraint languages should be considered.

4.2. *Classification of constraints according to the severity of constraint violations*

A concrete constraint is instantiated from one of the 81 constraint types and defined for a specific vocabulary. It does not make sense to determine the severity of constraint violations of an entire constraint type, as the severity depends on the individual context and vocabulary. SBE experts determined the default *severity level* for each of the 115 constraints to indicate how serious the violation of the constraint is. Although we provide default severity levels for each constraint, validation environments should enable users to adapt the severity levels of constraints according to their individual needs. The possibility to define severity levels for concrete constraints is in itself a requirement (*R-158*).

We use the classification system of log messages in software development like *Apache Log4j 2*, the *Java Logging API*, and the *Apache Commons Logging API* as many data practitioners also have experience in software development and software developers intuitively understand these levels. We simplify this commonly accepted classification system and distinguish the three severity levels (1) *informational*, (2) *warning*, and (3) *error*. Violations of *informational* constraints point to desirable

but not necessary data improvements to achieve RDF representations which are ideal in terms of syntax and semantics of used vocabularies. *Warnings* indicate syntactic or semantic problems which typically should not lead to an abortion of data processing. *Errors*, in contrast, are syntactic or semantic errors which should cause the abortion of data processing.

Note that there is indeed a correlation between the severity of a constraint and the classification of its type: *RDFS/OWL Based* constraints are in many cases associated with an *error* level as they typically represent basic constraints: there is a reason why they have been included in the vocabulary specification.

4.3. *Classification examples*

To get an overview on the constraint types contained in each of the three sets of constraint types, we delineate concrete constraints on the three vocabularies, group them by constraint type set, and classify them according to the severity of their violation.

RDFS/OWL Based. It is a common requirement to narrow down the value space of properties by an exhaustive enumeration of valid values (*R-30/37: allowed values*): *disco:CategoryStatistics*, e.g., can only have *disco:computationBase* relationships to the literals *"valid"* and *"invalid"* of the datatype *rdf:langString* (default severity level: *error*). Consider the following *DL knowledge base* $\mathcal{K}$, a collection of formal statements corresponding to *facts* or what is known explicitly:

$$\begin{aligned}\mathcal{K} = \{\ & \texttt{CategoryStatistics} \equiv \forall\ \texttt{computationBase}.\\ & \{\texttt{valid,invalid}\} \sqcap \texttt{langString},\\ & \texttt{Variable} \equiv \exists\ \texttt{concept.Concept},\\ & \texttt{DataSet} \sqsubseteq \forall\ \texttt{structure.DataStructureDefinition},\\ & \exists\ \texttt{hasTopConcept}.\top \sqsubseteq \texttt{ConceptScheme},\\ & \top \sqsubseteq \forall\ \texttt{variable.Variable}\ \}\end{aligned}$$

The constraint type *existential quantifications* (*R-86*) can be used to enforce that instances of given classes must have some property relation to individuals/literals of certain types. Variables, e.g., should have a relation to a theoretical concept (*informational*). The default severity level of this constraint is weak, as in most cases research can be continued without having information about the theoretical concept of a variable.

A *universal quantification* (*R-91*) contains all those individuals that are connected by a property only to individuals/literals of particular classes or data ranges. Resources of the type *qb:DataSet*, e.g., can only have *qb:structure* relationships to *qb:DataStructureDefinitions* (*error*).

Property domains (*R-25*) and *property ranges* (*R-35*) constraints restrict domains and ranges of properties: only *skos:ConceptSchemes*, e.g., can have *skos:hasTopConcept* relationships (*error*) and *disco:variable* relations can only point to *disco:Variables* (*error*).

It is often useful to declare a given (data) property as the *primary key* (*R-226*) of a class, so that a system can enforce uniqueness and build URIs from user inputs and imported data. In DDI-RDF, resources are uniquely identified by the property *adms:identifier*, which is therefore inverse-functional (`funct identifier`$^-$), i.e., for each *rdfs:Resource* x, there can be at most one distinct resource y such that y is connected by *adms:identifier*$^-$ to x (*error*). Keys, however, are even more general than *inverse-functional properties* (*R-58*), as a key can be a data property, an object property, or a chain of properties [24]. For this reason, as there are different sorts of key, and as keys can lead to undecidability, DL is extended with the construct *keyfor* (`identifier keyfor Resource`) [25] which is implemented by the OWL 2 *hasKey* construct.

Constraint Language Based. Depending on property datatypes, two different literal values have a specific ordering with respect to operators like $<$(*R-43: literal value comparison*). Start dates (*disco:startDate*), e.g., must be before ($<$) end dates (*disco:endDate*).

In many cases, resources must be *members of controlled vocabularies* (*R-32*). If a QB dimension property, e.g., has a *qb:codeList*, then the value of the dimension property on every *qb:Observation* must be in that code list (*error*).

Default values for objects (*R-31*) or literals (*R-38*) of given properties are inferred automatically when the properties are not present in the data. The value *true* for the property *disco:isPublic* indicates that a *disco:LogicalDataSet* can be accessed by anyone. Per default, however, access to data sets should be restricted (*false*) (*informational*).

SPARQL Based. The purpose of constraints of the type *data model consistency* is to ensure the integrity of the data according to the intended semantics of vocabularies' conceptual models about data. Every *qb:Observation*, e.g., must have a value for each dimension declared in its *qb:DataStructureDefinition* (*error*) and no two *qb:Observations* in the same *qb:DataSet* can have the same value for all dimensions (*warning*). If a *qb:DataSet* D has a *qb:Slice* S, and S has a *qb:Observation* O, then the *qb:DataSet* corresponding to O must be D (*warning*).

Objects/literals can be declared to be ordered for given properties (*R-121/217: ordering*). Variables, questions, and codes, e.g., are typically organized in a particular order. If codes (*skos:Concept*) should be ordered, they must be members (*skos:memberList*) in an ordered collection (*skos:Ordered Collection*) representing the code list of a variable (*informational*).

It is useful to declare properties to be *conditional* (*R-71*), i.e., if particular properties exist (or do not exist), then other properties must also be present (or absent). To get an overview on a study, either an abstract, a title, an alternative title, or links to external descriptions should be provided. If an abstract and an external description are absent, however, a title or an alternative title should be given (*warning*). In case a variable is represented in form of a code list, codes may

be associated with categories, i.e., human-readable labels (*informational*).

For data properties, it may be desirable to restrict that values of predefined languages must be present for determined number of times (*R-48/49: language tag cardinality*): (1) It is checked if literal language tags are set. Some controlled vocabularies, e.g., contain literals in natural language, but without information what language has actually been used (*warning*). (2) Language tags must conform to language standards (*error*). (3) Some thesaurus concepts are labeled in only one, others in multiple languages. It may be desirable to have each concept labeled in each of the languages that are also used on the other concepts, as language coverage incompleteness for some concepts may indicate shortcomings of thesauri (*informational*) [18].

5. Implementation

We claim that RDF data can be validated on constraints of each type expressed in any RDF-based language using SPARQL as low-level executing language. This claim is supported by the subsequent facts: [26] showed that constraints can be translated into non-recursive Datalog programs for validation, while [27] proved that SPARQL has the same expressive power as non-recursive Datalog programs. Therefore, data validation can be reduced to SPARQL query answering and SPARQL queries can be executed to validate RDF data against constraints of any type represented in any RDF-based language which has to be expressible in SPARQL.

Since the validation of each of the 81 constraint types can be implemented using SPARQL, we use *SPIN*, a SPARQL-based way to formulate and check constraints, as basis to develop a validation environment to validate RDF data according to constraints expressed in arbitrary constraint languages [1]. The *RDF Validator*, online available at http://purl.org/net/rdfval-demo, can directly be used to validate RDF data against constraints extracted from or defined for the three vocabularies DDI-RDF, QB, and SKOS. Additionally, own constraints on any vocabulary can be defined using several constraint languages.

The overall idea is that we see constraint languages as domain-specific languages, hence *domain-specific constraint languages (DSCL)*, that are translated into SPARQL and executed on RDF data within our validation environment we based on SPIN. Language designers are shifting attention from general purpose languages to domain-specific languages. *General-purpose languages* like Java and C++ for programming have been the primary focus of language research for a long time. The idea was to create only one language that is better suited for programming than any other language [28].

A *domain-specific language* [29] is a small, usually declarative language, tailored to a particular kind of problem [30] and offering substantial gains in expressiveness and ease of use compared with general purpose languages for the domain in question [31]. Instead of aiming to be the best for solving any kind of computing problem, domain-specific languages aim to be particularly good for solving a specific class of

problems, and in doing so they are often much more accessible to the general public than traditional general-purpose languages .

The translation of a DSCL into SPARQL queries is done once, for instance, by the designer of the DSCL, and provided in form of a *SPIN mapping* plus optional *pre-processing* instructions. From a user's perspective, all that is needed is a representation of *constraints* in the DSCL and some *data* to be validated against these constraints. All these resources are purely declarative and provided in RDF or as SPARQL queries. The actual execution of the constraint validation process is trivial using SPIN and illustrated in Figure 1.

First, an RDF graph has to be populated as follows:

(1) The *data* is loaded that is to be validated,
(2) the *constraints* in the DSCL are loaded,
(3) the *SPIN mapping* is loaded that contains the SPARQL representation of the DSCL, and
(4) the *pre-processing* is performed, which can be provided in form of SPARQL CONSTRUCT queries.

When the input RDF graph is ready, the SPIN engine checks for each resource if it satisfies all constraints defined in the DSCL and associated with the classes assigned to the resource, and generates a result RDF graph containing information about all constraint violations. With this framework, we have all we need to implement our own DSCL.

With this implementation, there are two obvious limitations of our approach: (1) constraints must be representable in RDF and (2) constraint languages must be expressible in SPARQL, i.e., the actual checking of constraints of each type the language supports must be expressible in form of a SPARQL query. For OWL 2, DSP, ShEx, ReSh, and SHACL, this is the case, but in the future, non-RDF based languages are expected to be supported as well.

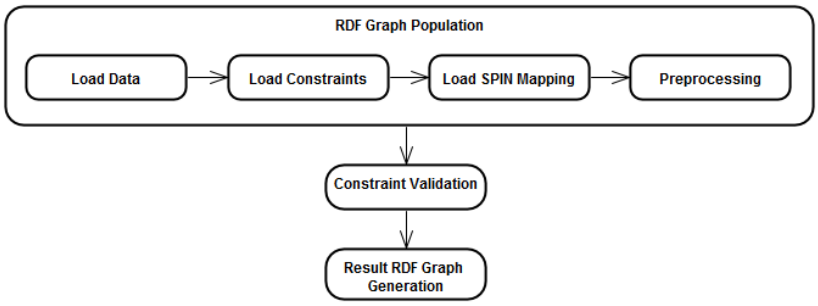

Fig. 1: Constraint Validation Process

5.1. *Connect SPIN to your data*

A SPIN mapping consists of multiple *SPIN construct templates* - each of them containing a SPARQL query executable to validate RDF data against constraints of a particular type. These templates are linked to the generic class *ToValidate* whose instances are validated on constraints of a certain type expressed in the DSCL for which the mapping is defined:

```
:ToValidate
    spin:constraint
        [ a dsp2spin:StatementTemplates_MinimumOccurrenceConstraint ] .
```

As the mapping is designed to be independent of any concrete data, the class *ToValidate* is purely generic. Instead of using such a generic class, it is also possible to link a template, responsible to check constraints of a given type, to the classes *owl:Thing* or *rdfs:Resource* to achieve that all instances of the input RDF graph are validated on constraints of that type.

Neither of these classes has to be assigned manually and explicitly to instances within the data to be validated. They are either implicitly inferred by means of reasoning or explicitly assigned during the *pre-processing* step, both in an automatic way. A reasonable approach would be to specify *ToValidate* as super-class of all classes whose instances should actually be validated against constraints (in the case of DSP: classes that are linked via `dsp:resourceClass` to a description template); this can be accomplished by a suitable SPARQL CONSTRUCT query that is executed before the validation starts. After *pre-processing*, the data might look like in the following code snippet – with the added assignment to the generic class in italics.

```
:Description-Logic-Handbook
    a :Computer-Science-Book , :ToValidate ;
    :subject "Computer Science" .
```

5.2. *Mapping from a DSCL to SPIN*

The mapping from a DSCL to SPIN is performed by creating SPIN construct templates - one for each constraint type that is supported by the DSCL, so for the constraint type *minimum unqualified cardinality restrictions (R-81)* expressible in DSP:

```
dsp2spin:StatementTemplates_MinimumOccurrenceConstraint
    a spin:ConstructTemplate ;
    spin:body [
        a sp:Construct ;
        sp:templates (...) ;
        sp:where (...) ] .
```

5.2.1. *Representing validation results as constraint violations in RDF*

This is the general structure of a SPIN construct template representing a SPARQL CONSTRUCT query in RDF. We use SPARQL CONSTRUCT queries to generate descriptions for each detected constraint violation:

```
CONSTRUCT {
    _:constraintViolation
        a spin:ConstraintViolation ;
        spin:violationRoot ?subject ;
        rdfs:label ?violationMessage ;
        spin:violationSource ?violationSource ;
        :severityLevel ?severityLevel ;
        spin:violationPath ?violationPath ;
        spin:fix ?violationFix }
```

In SPIN, the CONSTRUCT part of a SPARQL CONSTRUCT query is represented in RDF as follows:

```
a sp:Construct ;
sp:templates (
    [   sp:subject _:constraintViolation ;
        sp:predicate rdf:type ;
        sp:object spin:ConstraintViolation ]
    [   sp:subject _:constraintViolation ;
        sp:predicate rdfs:label ;
        sp:object [ sp:varName "violationMessage" ] ] ... ) ;
```

Representing constraint violations and therefore validation results in RDF enables to process them further by means of Semantic Web technologies. SPIN construct templates generate constraint violation triples indicating the subject, the properties, and the constraints causing the violations and the reason why violations have been raised.

A SPIN construct template creates constraint violation triples if all triple patterns within the SPARQL WHERE clause of the SPARQL CONSTRUCT query match. If we specify the existential quantification that a book must have at least one author and if the book *The-Hound-Of-The-Baskervilles* has no *author* relationship, all the triple patterns within the SPARQL WHERE clause match and the SPIN construct template for checking constraints of the type *existential quantifications (R-86)* generates a violation triple.

Constraint violations (*spin:constraintViolation*) should provide useful messages (*rdfs:label*) explaining the reasons why the data did not satisfy the constraint in order to assist data debugging and repair. In addition, constraint violation triples contain references to the subjects (*spin:violationRoot*), the properties (*spin:violationPath*), and the constraints (*spin:violationSource*) causing the violations. In the example, the subject *The-Hound-Of-The-Baskervilles*, the property *author*, and the *existential quantification* caused the violation.

Constraint violation triples may be linked to useful messages explaining how to overcome raised violations. To fix constraint violations (*spin:fix*), we may give some guidance how to become valid data either by adding, modifying, or deleting triples.

To indicate how severe the violation of a constraint is, we introduce a new property to classify constraints according to different levels of severity (*severityLevel*) like *informational*, *warning*, and *error*. It is also important to find not validated triples, i.e., triples which have not been validated against any constraint, as it may be enforced that every triple of the input graph has to be validated.

5.2.2. *Representing constraint checks in RDF*

One of the main benefits of SPIN is that arbitrary SPARQL queries and thus constraint checks are representable as RDF triples. SPIN provides a vocabulary, the *SPIN SPARQL Syntax*, to model SPARQL queries in RDF [32]. An RDF representation of constraint checks enables that (1) they can be consistently stored together with ontologies, constraints, and the data, (2) they can easily be shared on the Web of Data, (3) they can directly be processed by a plethora of already existing RDF tools, (4) they are linkable to constraints and RDF data, and (5) the validation of RDF data can automatically be executed by SPARQL execution engines.

As for the CONSTRUCT part of the SPARQL CONSTRUCT query, SPIN also represents the WHERE clause in RDF, i.e., the actual check if constraints of a given type hold for the data. The subsequent code snippet demonstrates how SPIN represents SPARQL 1.1 NOT EXISTS [33] filter expressions in RDF (`FILTER NOT EXISTS { ?book author ?person }`) using the RDF Turtle syntax:

```
[   a sp:Filter ;
    sp:expression [
        a sp:notExists ;
        sp:elements (
            [   sp:subject [ sp:varName "book" ] ;
                sp:predicate author ;
                sp:object [ sp:varName "person" ] ] ) ] ] )
```

As the mapping of a DSCL to SPIN is defined for all constraint types supported by the DSCL and hence independently of concrete constraints, constraints of all these types are generally checked for each instance of the generic class *ToValidate*. Therefore, the WHERE clause of a template always has to be restricted to classes for which concrete constraints are actually defined - in the case of DSP, the resource classes substituting the SPARQL variable *?resourceClass*:

```
WHERE { ?subject rdf:type ?resourceClass . }
```

6. Evaluation

In this section, based on an automatic constraint checking of a large amount of RDF data sets, we formulate several findings to gain valuable insights and make recommendations to direct the further development of constraint languages.

Despite the large volume of the evaluated data sets in general, we have to keep in mind that in this study we only validate data against constraints for three vocabularies. We took all well-established and newly developed SBE vocabularies into

account and defined constraints for the three vocabularies of them, which are most commonly used in the SBE sciences. For these three vocabularies, large data sets have already been published. For other SBE vocabularies, however, there is often not (yet) enough data openly available to draw general conclusions. Yet, these three vocabularies are representative, cover different aspects of SBE research data, and are also a mixture of widely adopted, accepted, and well-established vocabularies (QB and SKOS) on the one side and a vocabulary under development (DDI-RDF) on the other side.

As the evaluation is based on only three vocabularies, we cannot make valid general statements for all vocabularies, but we can formulate several findings and make recommendations to direct the further development of constraint languages. As these findings cannot be proved yet, they still have to be verified or falsified by evaluating the quality of RDF data represented by further well-established and newly developed vocabularies - used within the SBE sciences and other domains.

6.1. *Experimental setup*

On the three vocabularies DDI-RDF, QB, and SKOS, we collected 115 constraints,[a] either from the vocabularies themselves or from several domain experts, and classified and implemented them for data validation. We ensured that these constraints are equally distributed over the sets and vocabularies we have. We then evaluated the data quality of 15,694 data sets (4.26 billion triples) of SBE research data, obtained from 33 SPARQL endpoints, by validating the data against these 115 constraints. Table 1 lists the number of validated data sets and the overall sizes in terms of validated triples for each of the vocabularies.

Table 1: Number of Validated Data Sets and Triples for each Vocabulary

Vocabulary	Data Sets	Triples
QB	9,990	3,775,983,610
SKOS	4,178	477,737,281
DDI-RDF	1,526	9,673,055

We validated, i.a., (1) QB data sets published by the *Australian Bureau of Statistics*, the *European Central Bank*, and the *Organisation for Economic Co-operation and Development*, (2) SKOS thesauri like the *AGROVOC Multilingual agricultural thesaurus*, the *STW Thesaurus for Economics*, and the *Thesaurus for the Social Sciences*, and (3) DDI-RDF data sets provided by the *Microdata Information System*, the *Data Without Boundaries Discovery Portal*, the *Danish Data Archive*, and the *Swedish National Data Service*. In a technical report, we describe the evaluation in further detail for each vocabulary, queried SPARQL endpoint, and constraint [34].

[a]Implementations of all 115 constraints are online available at: https://github.com/github-thomas-hartmann/phd-thesis/tree/master/chapter/chapter-9/constraints

Furthermore, we have published the evaluation results for each QB data set in form of one document per SPARQL endpoint.[b]

6.2. *Evaluation results and formulation of findings*

Tables 2 and 3 show the results of the evaluation, more specifically the numbers of constraints and constraint violations, which are caused by these constraints, in percent; whereas the numbers in the first line indicate the absolute amount of constraints and violations. The constraints and their raised violations are grouped by vocabulary, which type of language the types of the constraints are formulated with, and their severity level.

The numbers of validated triples and data sets differ between the vocabularies as we validated 3.8 billion QB, 480 million SKOS, and 10 million DDI-RDF triples. To be able to formulate findings which apply for all vocabularies, we only use normalized relative values representing the percentage of constraints and violations belonging to the respective sets.

There is a strong overlap between *RDFS/OWL* and *Constraint Language Based* constraint types as in many cases constraint types are expressible by RDFS/OWL and classical constraint languages. This is the reason why the percentage values of constraints and violations grouped by the classification of constraint types according to the expressivity of constraint language types do not accumulate to 100%.

Table 2: Evaluation Results (1)

	DDI-RDF		**QB**	
	C	CV	C	CV
	78	3,575,002	20	45,635,861
SPARQL	29.5	34.7	**60.0**	**100.0**
CL	**64.1**	**65.3**	40.0	0.0
RDFS/OWL	**66.7**	**65.3**	40.0	0.0
info	**56.4**	**52.6**	0.0	0.0
warning	11.5	29.4	15.0	**99.8**
error	32.1	18.0	**85.0**	0.3

C (constraints), CV (constraint violations)

Almost 2/3 of all constraints, nearly 1/3 of the DDI-RDF, 60% of the QB, and all SKOS constraints are *SPARQL Based*. For well-established vocabularies, the most formulated constraints are *SPARQL Based* (80%). For newly developed vocabularies, however, the most expressed constraints are *RDFS/OWL Based* (2/3). Nearly 80% of all violations are caused by *SPARQL*, 1/5 by *Constraint Language*, and 1/5 by *RDFS/OWL Based* constraints.

Finding 1 *The facts that 80% of all violations are raised by SPARQL Based constraints and that 2/3 of all constraints are SPARQL Based, increases the importance*

[b]Online available at: https://github.com/github-thomas-hartmann/phd-thesis/tree/master/chapter/chapter-9/evaluation/data-sets/QB

Table 3: Evaluation Results (2)

	SKOS		***Total***	
	C	CV	C	CV
	17	5,540,988	115	54,751,851
SPARQL	**100.0**	**100.0**	**63.2**	**78.2**
CL	0.0	0.0	34.7	21.8
RDFS/OWL	0.0	0.0	35.6	21.8
info	**70.6**	41.2	**42.3**	31.3
warning	29.4	**58.8**	18.7	**62.7**
error	0.0	0.0	**39.0**	6.1

C (constraints), CV (constraint violations)

to formulate constraints, which up to now can only be expressed in SPARQL, using high-level constraint languages. Data quality can be significantly improved when suitable constraint languages are developed which enable to define SPARQL Based constraints in an easy, concise, and intuitive way. Thereby, the more elaborate a vocabulary is, the more sophisticated and complex constraints are specified using SPARQL.

These constraints are of such complexity that up to now in most cases they can only be expressed by plain SPARQL. It should be an incentive for language designers to devise languages which are more intuitive than SPARQL in a way that also domain experts, which are not familiar with SPARQL, can formulate respective constraints.

Finding 2 *The fact that only 1/5 of all violations result from RDFS/OWL Based constraints, even though more than 1/3 of all constraints are RDFS/OWL Based, indicates good data quality for all vocabularies with regard to their formal specifications.*

Finding 3 *As more than 1/3 of all constraints are RDFS/OWL Based, the first step to make progress in the further development of constraint languages is to cover the constraint types which can already be formulated using RDFS and OWL.*

While 2/3 of the DDI-RDF violations result from *RDFS/OWL Based* constraints, QB and SKOS violations are only raised by *SPARQL Based* constraints.

Finding 4 *For well-established vocabularies, RDFS/OWL Based constraints are almost completely satisfied which generally indicates very impressive data quality, at least in the SBE domain and for the basic requirements. For newly developed vocabularies, however, data quality is poor as RDFS/OWL Based constraints are not fulfilled.*

For DDI-RDF, data providers still have to understand the vocabulary and of course data cannot have high quality if the specification is not yet stable. It is likely that a newly developed vocabulary is still subject of constant change and that early

adopters did not properly understand its formal specification. Thus, published data may not be consistent with the current draft of its conforming vocabulary.

When vocabularies under development turn into well-established ones, data providers are experienced in publishing their data in conformance with these vocabularies and formal specifications are more elaborated. As a consequence, *RDFS/OWL Based* constraints are satisfied to a greater extend which leads to better data quality.

The reason why we only defined *SPARQL Based* constraints on SKOS for assessing the quality of thesauri is that literature and practice especially concentrate on evaluating graph-based structures of thesauri by applying graph- and network-analysis techniques which are of such complexity that they can only be implemented in SPARQL.

Almost 40% of all constraints are *error*, more than 40% are *informational*, and nearly 20% are *warning* constraints. *Informational* constraints caused approximately 1/3 and *warning* constraints narrowly 2/3 of all violations.

Finding 5 *Although 40% of all constraints are error constraints, the percentage of severe violations is very low, compared to about 2/3 of warning and 1/3 of informational violations. This implies that data quality is high with regard to the severity level of constraints and that proper constraint languages can significantly improve data quality beyond fundamental requirements.*

We did not detect violations of *error* constraints for well-established vocabularies, even though 85% of the QB constraints are *error* constraints. More than 50% of the DDI-RDF constraints and more than 70% of the SKOS constraints are *informational* constraints. 1/6 of the DDI-RDF violations are caused by *error* constraints and almost all QB and 59% of the SKOS violations are caused by *warning* constraints.

Finding 6 *For well-established vocabularies, data quality is high as serious violations rarely appear (0.3% for QB). For newly developed vocabularies, however, data quality is worse as serious violations occur partially (1/6 for DDI-RDF).*

Especially for vocabularies under development, constraint languages should be used to a larger extend in addition to RDFS/OWL in order to define appropriate constraints to detect and solve severe violations.

80% of the violations, which are raised by either *RDFS/OWL* or *Constraint Language Based* constraints, are caused by constraints with the severity level *informational* (see Table 4) and almost all (94%) of the violations, which are caused by *SPARQL Based* constraints, are raised by *warning* constraints. Approximately 1/2 of all constraints are *informational* constraints regardless how their types are classified according to the expressivity of constraint language types.

Table 4: Constraints and Violations by Language Type and Severity

	RDFS/OWL		***CL***		***SPARQL***	
	C	CV	C	CV	C	CV
info	**52.5**	**79.64**	**55.2**	**79.60**	**45.1**	4.39
warning	18.0	20.28	15.5	20.27	19.6	**94.17**
error	29.5	0.08	29.3	0.13	35.3	1.43

C (constraints), CV (constraint violations)

Finding 7 *Whatever language is used to formulate constraints, 1/2 of all constraints are informational, 1/3 are error, and 1/5 are warning constraints.*

The fact, that regardless of the language used to specify constraints, 1/2 of all constraints are *informational* indicates the importance that constraint languages support constraints on multiple levels. Constraints are by far not only used to prevent certain usages of a vocabulary, they are rather needed to provide better guidance for improved interoperability.

Finding 8 *Regardless of the type of the used language, there are only a few violations raised by error constraints which stands for good data quality in general. In contrast, constraints of low severity, expressible by RDFS/OWL or high-level constraint languages, are violated to a large extent (80%), whereas more serious warning constraints, only expressible by SPARQL, are violated to an even larger extend (94%).*

94% of the violations caused by *SPARQL Based* constraints are *warnings*, which means that data quality could be significantly improved when solving these quite serious violations. We claim that this is more likely when these *SPARQL Based* constraints are not only expressible by plain SPARQL but also by high-level constraint languages enabling to formulate such constraints in a more intuitive and concise way.

7. Conclusion and Future Work

We captured 115 constraints on three vocabularies commonly used within the social, behavioral, and economic (SBE) sciences (DDI-RDF, QB, and SKOS), either from the vocabularies themselves or from several domain experts, and classified them based on by which of the three types of constraint languages (*RDFS/OWL Based*, *Constraint Language Based*, and *SPARQL Based*) their constraint types can be expressed.

In addition, we let the domain experts classify each constraint according to the severity of its violation. Although we provide default severity levels for each constraint, validation environments should enable users to adapt the severity levels of constraints according to their individual needs. We simplify the classification system of log messages in software development and differentiate between the three severity levels *informational*, *warning*, and *error*.

By validating against these constraints, we evaluated the data quality of 15,694 data sets (4.26 billion triples) of SBE research data, obtained from 33 SPARQL endpoints. This way, we gain a better understanding about the role of certain constraint types for assessing the quality of RDF data and therefore the usability of identified constraint types for determining RDF data quality.

Based on the evaluation results, we formulated several findings to direct the further development of constraint languages. The general applicability of these findings, however, is still to be confirmed beyond the examined vocabularies and for other domains. The main findings are:

(1) *Data quality can be significantly improved when suitable constraint languages are developed enabling to define constraints - which up to now can only be expressed by plain SPARQL - in an easy, concise, and intuitive way. Thereby, the more elaborate a vocabulary is, the more sophisticated and complex constraints are necessary, which in most cases can only be specified in SPARQL.*
(2) *As only 1/5 of all violations result from constraints which are expressible in RDFS or OWL, even though 1/3 of all constraints are representable in RDFS/OWL, data quality is high for all vocabularies with regard to their formal specifications.*
(3) *Although 40% of all constraints are associated with the severity level error, the percentage of severe violations is very low - compared to about 2/3 of warning and 1/3 of informational violations. This implies that data quality is high with regard to the severity of constraints and that proper constraint languages can significantly improve data quality beyond fundamental requirements.*
(4) *Whatever language is used to formulate constraints, 1/2 of all constraints are informational, 1/3 are error, and 1/5 are warning constraints. This fact emphasizes the importance that constraint languages support multiple levels of severity.*
(5) *Violations caused by constraints representable by RDFS/OWL or high-level constraint languages are of low severity, whereas the violation of constraints, only expressible in SPARQL, is more serious. This is the reason why there is a significant demand for high-level constraint languages that support the expression of constraints which up to now can only be formulated by plain SPARQL.*

We have been really impressed by the high quality of the QB and SKOS data. This is in contrast to the sometimes heard rumor that Linked Open Data lacks quality. We are actively involved in the further development and implementation of constraint languages and will use the results presented in the paper to set priorities on features where we expect the highest impact on the data quality of real-life data in the SBE domain. As the use of constraint languages per se enhances data quality, it must be continued working intensively on their further development.

The major differences between this journal paper and the extended conference paper [35] are that we (1) give an in-depth description of common vocabularies in the SBE sciences, (2) delineate a complete running example how SBE sciences data and metadata is represented in RDF, and (3) describe in detail how we implemented

our validation environment.

References

[1] T. Bosch and K. Eckert, "Towards description set profiles for rdf using sparql as intermediate language," in *Proc. of the 14th DCMI Int. Conf. on Dublin Core and Metadata Applications*, Austin, Texas, USA, 2014.

[2] ——, "Requirements on rdf constraint formulation and validation," in *Proc. of the 14th DCMI Int. Conf. on Dublin Core and Metadata Applications*, Austin, Texas, USA, 2014.

[3] R. Cyganiak and D. Reynolds, "The rdf data cube vocabulary," W3C, W3C Recommendation, 2014.

[4] R. Cyganiak, S. Field, A. Gregory, W. Halb, and J. Tennison, "Semantic statistics: Bringing together sdmx and scovo," in *Proc. of the Int. World Wide Web Conference, Workshop on Linked Data on the Web*, C. Bizer, T. Heath, T. Berners-Lee, and M. Hausenblas, Eds., 2010.

[5] Cyganiak, Richard and Dollin, Chris and Reynolds, Dave, "Expressing statistical data in rdf with sdmx-rdf," Tech. Rep., 2010.

[6] M. Hausenblas, D. Ayers, L. Feigenbaum, T. Heath, W. Halb, and Y. Raimond, "The statistical core vocabulary (scovo)," Digital Enterprise Research Institute (DERI), DERI Specification, 2012.

[7] M. Hausenblas, W. Halb, Y. Raimond, L. Feigenbaum, and D. Ayers, "Scovo: Using statistics on the web of data," in *Proc. of the 6th European Semantic Web Conf. on The Semantic Web: Research and Applications*.

[8] D. Vrandecic, C. Lange, M. Hausenblas, J. Bao, and L. Ding, "Semantics of governmental statistics data," in *Proc. of the 2nd Int. ACM Web Science Conf.*

[9] J. Wackerow, L. Hoyle, and T. Bosch, "Physical data description," DDI Alliance, DDI Alliance Specification, 2016.

[10] M. Vardigan, P. Heus, and W. Thomas, "Data documentation initiative: Toward a standard for the social sciences," *Int. Journal of Digital Curation*, vol. 3, no. 1, pp. 107 – 113, 2008.

[11] T. Bosch, R. Cyganiak, J. Wackerow, and B. Zapilko, "Ddi-rdf discovery vocabulary: A vocabulary for publishing metadata about data sets (research and survey data) into the web of linked data," DDI Alliance, DDI Alliance Specification, 2016.

[12] A. Miles and S. Bechhofer, "Skos simple knowledge organization system reference," W3C, W3C Recommendation, 2009.

[13] A. Isaac and E. Summers, "Skos simple knowledge organization system primer," W3C, W3C Working Group Note, 2009.

[14] A. Miles and S. Bechhofer, "Skos simple knowledge organization system extension for labels (skos-xl)," W3C, W3C Recommendation, 2009.

[15] F. Cotton, "Xkos - a skos extension for representing statistical classifications," DDI Alliance, DDI Alliance Specification, 2015.

[16] F. Cotton, R. Cyganiak, R. Grim, D. W. Gillman, Y. Jaques, and W. Thomas, "Xkos: An skos extension for statistical classifications," in *Proc. of the 59th World Statistics Congress of the Int. Statistical Institute*, The Hague, The Netherlands, 2013.

[17] D. W. Gillman, F. Cotton, and Y. Jaques, "extended knowledge organization system (xkos)," United Nations Economic Commission for Europe (UNECE), Geneva, Switzerland, Work Session on Statistical Metadata, 2013.

[18] C. Mader, B. Haslhofer, and A. Isaac, "Finding quality issues in skos vocabularies," in *Proc. of the Second Int. Conf. on Theory and Practice of Digital Libraries*. Berlin,

Heidelberg: Springer-Verlag, 2012, pp. 222–233.

[19] B. Motik, I. Horrocks, and U. Sattler, "Adding integrity constraints to owl," in *Proc. of the OWLED 2007 Workshop on OWL: Experiences and Directions*, vol. 258, Innsbruck, Austria, 2007.

[20] ——, "Bridging the gap between owl and relational databases," *Journal of Web Semantics*, vol. 7, no. 2, pp. 74–89, 2009.

[21] J. Tao, E. Sirin, J. Bao, and D. L. McGuinness, "Integrity constraints in owl," in *Proc. of the 24th AAAI Conf. on Artificial Intelligence*, Atlanta, Georgia, USA, 2010.

[22] P. F. Patel-Schneider, "Using description logics for rdf constraint checking and closed-world recognition," in *Proc. of the 29th AAAI Conf. on Artificial Intelligence*, Austin Texas, USA, 2015.

[23] T. Hartmann, B. Zapilko, J. Wackerow, and K. Eckert, "Constraints to validate rdf data quality on common vocabularies in the social, behavioral, and economic sciences," *Computing Research Repository (CoRR)*, vol. abs/1504.04479, 2015.

[24] M. Schneider, "Owl 2 web ontology language rdf-based semantics," W3C, W3C Recommendation, 2009.

[25] C. Lutz, C. Areces, I. Horrocks, and U. Sattler, "Keys, nominals, and concrete domains," *Journal of Artificial Intelligence Research*, vol. 23, no. 1, pp. 667–726, 2005.

[26] E. Sirin and J. Tao, "Towards integrity constraints in owl," in *Proc. of the 6th Int. Workshop on OWL: Experiences and Directions, 8th Int. Semantic Web Conf.*

[27] R. Angles and C. Gutierrez, "The expressive power of sparql," in *The Semantic Web - ISWC 2008*, ser. Lecture Notes in Computer Science, A. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. Finin, and K. Thirunarayan, Eds. Springer Berlin Heidelberg, 2008, vol. 5318, pp. 114–129.

[28] W. Taha, "Domain-specific languages," in *Proc. of the 15th Int. Conf. on Computer Engineering and Systems*, Houston, TX, USA, 2008.

[29] M. Fowler, *Domain-Specific Languages*, ser. Addison-Wesley Signature Series. Pearson Education, 2010.

[30] A. Raja and D. Lakshmanan, "Domain specific languages," *Int. Journal of Computer Applications*, vol. 1, no. 21, pp. 99–105, 2010.

[31] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," *ACM Computing Surveys*, vol. 37, no. 4, pp. 316–344, 2005.

[32] H. Knublauch, "SPIN - SPARQL Syntax," W3C, W3C Member Submission, February 2011, http://www.w3.org/Submission/2011/SUBM-spin-sparql-20110222/.

[33] S. Harris and A. Seaborne, "Sparql 1.1 query language," W3C, W3C Recommendation, 2013.

[34] T. Hartmann, B. Zapilko, J. Wackerow, and K. Eckert, "Evaluating the quality of rdf data sets on common vocabularies in the social, behavioral, and economic sciences," *Computing Research Repository (CoRR)*, vol. abs/1504.04478, 2015.

[35] ——, "Validating rdf data quality using constraints to direct the development of constraint languages," in *Proc. of the 10th Int. Conf. on Semantic Computing.*