Classification of RDF Constraint Types to Ensure High Quality of Metadata and Data

Thomas Bosch[1], Benjamin Zapilko[1], Joachim Wackerow[1], and Kai Eckert[2]

[1] GESIS – Leibniz Institute for the Social Sciences, Germany
`{firstname.lastname}@gesis.org`,
[2] Stuttgart Media University, Germany
`eckert@hdm-stuttgart.de`

Abstract. For research institutes, data libraries, and data archives, RDF data validation according to predefined constraints is a much sought-after feature, particularly as this is taken for granted in the XML world. To ensure high quality and trust, both metadata and data must satisfy certain criteria - specified in terms of RDF constraints.

The **contribution** of this paper is the development of a system (1) to classify RDF constraints (according to their severity levels) to evaluate the quality of (meta)data which may be represented by any vocabulary and (2) to classify RDF constraint types (according to their complexity) which in most cases correspond to RDF validation requirements. By defining a huge amount of constraints of the majority of constraint types to validate data sets from the community around research data for the *social, behavioural, and economic (SBE) sciences*, we prove the claim that the proposed classification system perfectly applies vocabulary-independent and therefore we prove its generality.

We evaluated the (meta)data quality of large real world data sets (more than 4.2 billion triples and 15 thousand data sets) from the *SBE* domain represented by multiple vocabularies to get an understanding (1) which sets of constraint types (on different levels of complexity) and (2) which sets of constraints (associated with particular severity levels) encompass the constraints causing the most/fewest constraint violations.

Keywords: RDF Validation, RDF Constraints, Metadata Quality, Data Quality, Tabular Data, DDI-RDF Discovery Vocabulary, RDF Data Cube Vocabulary, SKOS, Thesauri, Linked Data, Semantic Web

1 Introduction

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)*, an XML format to disseminate, manage, and reuse data collected and archived for research [11]. Increasingly, data professionals, data archives, data libraries, government statisticians (e.g. data.gov, data.gov.uk), and national statistical institutes are very interested in having their data be discovered and used by exposing their metadata (e.g. about unemployment rates or income) within the Web of Linked Data. As *DDI* XML documents are validated against diverse XSDs[3] (see section 10), the *SBE* community is used to have this functionality in the XML world. Several languages exist to formulate and check constraints on RDF data, ranging from *Shape Expressions*, *Resource Shapes* or *Description Set Profiles* to using *OWL 2* as a constraint language and the SPARQL-based *SPIN*[4]. In 2013, the W3C organized the RDF Validation Workshop[5], where experts from industry, government, and academia discussed first use cases for RDF 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[6] and the DCMI RDF Application Profiles task group[7].

Bosch and Eckert [1] collected the findings of these working groups and initiated a database of RDF validation requirements which is available for contribution at http://purl.org/net/rdf-validation. The intention is to collaboratively collect case studies, use cases, requirements, and solutions regarding RDF validation in a comprehensive and structured way. The requirements are classified to better evaluate existing solutions and each requirement is directly mapped to a constraint type which is expressible by at least one constraint language. As *SPIN* is the within the RDF validation community accepted and widely adopted way to check constraints on RDF data, Bosch and Eckert [2] use *SPIN* as basis to define a validation environment (available at http://purl.org/net/rdfval-demo) in which the validation of any constraint language[8] can be implemented by representing them in SPARQL. The SPIN engine checks for each resource if it satisfies all constraints, which are associated with its assigned classes, and generates a result RDF graph containing information about all constraint violations.

The **contribution** of this paper is the development of a system

1. to classify RDF constraints (according to their severity levels) to evaluate the quality of metadata and data which may be represented by any vocabulary and
2. to classify RDF constraint types (according to their complexity) which in most cases correspond to RDF validation requirements[9] (sections 4 - 7).

By defining a huge amount of constraints of the majority of the constraint types, we apply the developed classification system to several and different vocabularies from the *SBE* domain to represent both metadata and data and therefore prove

[3] http://www.ddialliance.org/Specification/
[4] http://spinRDF.org/
[5] http://www.w3.org/2012/12/rdf-val/
[6] http://www.w3.org/2014/rds/charter
[7] http://wiki.dublincore.org/index.php/RDF-Application-Profiles
[8] The only limitation is that constraint languages must be represented in RDF
[9] For simplicity reasons, we use the terms *constraint types* and *constraints* instead of *RDF constraint types* and *RDF constraints* in the rest of the paper

its generality. A complex and complete real world running example from the *SBE* domain serves to prove the claim that the developed classification system perfectly applies for diverse vocabularies. We describe why RDF validation is important for the *SBE* community (section 2), how data in tabular format and metadata on person-level data sets, aggregated data sets, and thesauri are represented in RDF, how therefore reused vocabularies are interrelated (section 3), and how *SBE* (meta)data is validated against constraints of different constraint types (sections 4 - 7). We evaluated the (meta)data quality of large real world data sets (more than 4.2 billion triples and 15 thousand data sets) from the *SBE* domain represented by multiple vocabularies to get an understanding (1) which sets of constraint types (on different levels of complexity) and (2) which sets of constraints (associated with particular severity levels) encompass the constraints causing the most/fewest constraint violations (see section 9).

2 Motivation

The data most often used in research within the *SBE* community is *person-level data* (or more generally *record-unit 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). The range of person-level data is very broad - including census, education, 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.

By its nature, person-level 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 person-level 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 person-level data. Aggregated data is more and more published in form of CSV files, allowing to perform data calculations. Portals harvest metadata (as well as publicly available data) from multiple RDF data providers. To ensure high quality, (meta)data must satisfy certain criteria - specified in terms of RDF constraints.

For more detailed analyses, researchers refer to person-level data including additional variables needed to answer subsequent research questions like the comparison of studies between countries. A *study* represents the process by which a data set was generated or collected. Eurostat[10], the statistical office of the European Union, provides research findings in form of aggregated data (downloadable as CSV files) and its metadata at European level that enable comparisons

[10] http://ec.europa.eu/eurostat

between countries. The variable *formal childcare*[11] captures the measured availability of childcare services in percent over the population in European Union member states by the variables *year*, *duration* (in hours per week), *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').

To determine if variables measuring *age* - collected for different countries (age_{DE}, age_{UK}) - are comparable, diverse constraints are checked: (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 over comparable 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 last constraint.

3 Vocabularies to Represent Metadata and Data in RDF

The *RDF Data Cube Vocabulary (QB)*[12] is a W3C recommendation for representing metadata on *data cubes*, i.e. multi-dimensional aggregated data, in RDF [6]. A *qb:DataStructureDefinition* contains metadata of the data collection. The variable *formal childcare* is modelled as *qb:measure*, since it stands for what has been measured in the data collection. The variables *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* which contains values for each dimension[13].

Physical Data Description (PHDD)[14] is a vocabulary to represent data in tabular format in RDF enabling further aggregations and calculations. The data could be either represented in records with character-separated values (CSV) or fixed length. *Eurostat* provides a CSV file, a two-dimensional table (*phdd:Table*) about *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

[11] Aggregated data and its metadata is available at: http://ec.europa.eu/eurostat/web/products-datasets/-/ilc_caindformal
[12] http://www.w3.org/TR/vocab-data-cube/
[13] The complete running example in RDF is available at: https://github.com/boschthomas/rdf-validation/tree/master/data/running-example
[14] https://github.com/linked-statistics/physical-data-description

percent, the column position (*5*), the recommended data type (*xsd:nonNegativeInteger*), and the storage format (*TINYINT*) is stated.

For more detailed analyses we refer to the metadata on person-level data collected for the series *EU-SILC (European Union Statistics on Income and Living Conditions)*[15]. Where data collection is cyclic, data sets may be released as *series*, where each cycle produces one or more data sets. Aggregated (qb:DataSet) and underlying person-level data sets (*disco:LogicalDataSet*) are connected by *prov:wasDerivedFrom*. The aggregated variable *formal childcare* is calculated on the basis of six person-level variables (e.g., *Education at pre-school*[16]) for which detailed metadata is given (e.g., code lists) enabling researchers to replicate the results shown in aggregated data tables. The *DDI-RDF Discovery Vocabulary (Disco)* is a vocabulary to represent metadata on person-level data in RDF. The series (*disco:StudyGroup*) *EU-SILC* contains one study (*disco:Study*) for each year (*dcterms:temporal*) of data collection. *dcterms:spatial* points to the countries for which the data has been collected. The study *EU-SILC 2011* contains eight person-level data sets (*disco:LogicalDataSet*) including person-level variables (*disco:Variable*) like the six ones needed to calculate the aggregated variable *formal childcare*.

The *Simple Knowledge Organization System (SKOS)* is reused multiple times to build *SBE* vocabularies. The codes of the variable *Education at pre-school* (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*). *skos:narrower* builds the hierarchy of theoretical concepts within a *skos:ConceptScheme* of a series. 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.

4 Classification of Constraint Types and Constraints

Bosch et al. identified 76 requirements to formulate RDF constraints (e.g. *R-75: minimum qualified cardinality restrictions*); each of them corresponding to an RDF constraint type[17][3]. We published a technical report[18] in which we explain each requirement/constraint type in detail and give examples for each expressed by different constraint languages. The knowledge representation formalism *Description logics (DL)*, with its well-studied theoretical properties, provides the

[15] http://www.gesis.org/missy/eu/metadata/EU-SILC

[16] http://www.gesis.org/missy/eu/metadata/EU-SILC/2011/Cross-sectional/original#2011-Cross-sectional-RL010

[17] Constraint types and constraints are uniquely identified by alphanumeric technical identifiers like *R-71-CONDITIONAL-PROPERTIES*

[18] Available at: http://arxiv.org/abs/1501.03933

foundational basis for each constraint type. Therefore, this technical report contains mappings to *DL* to logically underpin each requirement and to determine which *DL* constructs are needed to express each constraint type [3].

We developed a system to classify both RDF constraint types and RDF constraints to evaluate the quality of metadata and data which may be represented by any vocabulary. We recently published a technical report[19] (serving as first appendix of this paper) in which we describe 213 constraints of 53 distinct constraint types to validate tabular data (*PHDD*) and metadata on person-level data sets (*Disco*), aggregated data sets (*QB*), and thesauri (*SKOS*). By applying the proposed classification system to several vocabularies to represent both metadata and data, we prove its generality [5].

4.1 Classification of Constraint Types

According to the complexity of constraint types, the complete set of *constraint types* ($\mathcal{CT}$) encompasses three disjoint *sets of constraint types*:

1. $\mathcal{CT}_B$: ***Basic Constraint Types***
2. $\mathcal{CT}_S$: ***Simple Constraint Types***
3. $\mathcal{CT}_C$: ***Complex Constraint Types***

The modeling languages *RDF*, *RDFS*, and *OWL* are typically used to define vocabularies. *Basic constraint types* ($\mathcal{CT}_B$) denotes the set of constraint types whose constraints can be extracted completely automatically out of vocabularies. As vocabularies have been specified using *RDF*, *RDFS*, and *OWL*, *basic constraints* ensure that the data is consistent with the intended syntaxes, semantics, and integrity of vocabularies' data models. *Minimum qualified cardinality restrictions* (*R-74*), 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. This way, it is expressible in *OWL 2* that a *phdd:TableStructure* has (*phdd:column*) at least one *phdd:Column*:

```
[   a owl:Restriction ; rdfs:subClassOf TableStructure ;
    owl:minQualifiedCardinality 1 ;
    owl:onProperty column ;
    owl:onClass Column ] .
```

Simple constraint types ($\mathcal{CT}_S$) is the set of constraint types whose constraints can be easily defined without much effort in addition to $\mathcal{CT}_B$ constraints. *Data property facets* (*R-46*) is an example of an $\mathcal{CT}_S$ constraint type which enables to declare frequently needed facets for data properties in order to validate input against simple conditions including min/max values, regular expressions, and string length. The abstract of series/studies, e.g., should have a minimum length.

Complex constraint types ($\mathcal{CT}_C$) encompass constraint types for which the definition of constraints is rather complex and cannot be derived from vocabulary definitions. Complex constraints show the importance of constraint languages

[19] Available at: http://arxiv.org/abs/1504.04479

enabling to describe more complex constraints. For assessing the quality of thesauri, e.g., we concentrate on the 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.

4.2 Classification of Constraints

SBE experts determined the default ***severity level*** (*R-158*) for each constraint to indicate how serious the violation of the constraint is. We propose an extensible metric to measure the continuum of severity levels ranging from $\mathcal{SL}_0$ to $\mathcal{SL}_2$. According to the constraints' default severity level the complete set of constraints ($\mathcal{C}$) encompasses three disjoint *sets of constraints*:

- $\mathcal{SL}_0$: set of constraints with **severity level *informational***
- $\mathcal{SL}_1$: set of constraints with **severity level *warning***
- $\mathcal{SL}_2$: set of constraints with **severity level *error***

Violations of $\mathcal{SL}_0$ constraints point to desirable data improvements to achieve RDF representations which are ideal in terms of syntax and semantics of used vocabularies. Data not conforming to $\mathcal{SL}_1$ and $\mathcal{SL}_2$ constraints is syntactically and/or semantically not correctly represented. The difference between $\mathcal{SL}_1$ and $\mathcal{SL}_2$ constraints is that data, not conforming to $\mathcal{SL}_1$ but conforming to $\mathcal{SL}_2$ constraints, could be processed further, whereas data, not corresponding to $\mathcal{SL}_2$ constraints, cannot be processed further after validation. As constraints of $\mathcal{CT}_B$ constraint types are derived from explicitly stated semantics of vocabularies, their default severity levels are in most cases very strong ($\mathcal{SL}_2$) and in average stronger than the severity levels of constraints assigned to $\mathcal{CT}_S$ and $\mathcal{CT}_C$ constraint types. As a consequence, violating many constraints of $\mathcal{CT}_B$ constraint types is an indicator for bad (meta)data quality[20].

Although, we provide default severity levels for each constraint, validation environments should enable users to adapt constraints' severity levels according to their individual needs. Validation environments should enable users to select which constraints to validate against depending on their individual use cases. For some use cases, validating constraints of $\mathcal{CT}_B$ constraint types may be more important than validating constraints of $\mathcal{CT}_S$ or $\mathcal{CT}_C$ constraint types. For other use cases, validating $\mathcal{SL}_2$ constraints may be sufficient without taking $\mathcal{SL}_1$ and $\mathcal{SL}_0$ constraints into account. We evaluated the (meta)data quality of large real world data sets represented by multiple and different vocabularies to get an understanding (1) which sets of constraint types (on different levels of complexity) and (2) which sets of constraints (associated with particular severity levels) encompass the constraints causing the most/fewest constraint violations (see section 9).

[20] For simplicity reasons, we only assign severity levels to $\mathcal{CT}_B$ constraints in this paper in case they differ from $\mathcal{SL}_2$.

5 Basic Constraint Types

As *RDFS* and *OWL* are typically used to define vocabularies, *RDFS* and *OWL* reasoning may be performed prior to validation. Reasoning and validation are indeed very closely related. *Reasoning* is the process of determining what follows from what has been stated. We divide the whole set of *basic constraint types* ($\mathcal{CT}_B$) into two disjoint sets to investigate the affect of reasoning to the validation process:

1. $\mathcal{C}_B^{\mathcal{R}}$ corresponds to axioms in *OWL 2* and denotes the set of constraint types which enable performing reasoning prior to validation, especially when not all the knowledge is explicit (section 5.1).
2. $\overline{\mathcal{C}_B^{\mathcal{R}}}$ denotes the set of constraint types for which reasoning cannot be done or does not improve the result in any obvious sense (section 5.2).

5.1 Basic Constraint Types with Reasoning

Validation environments should enable users to decide if they wish to perform reasoning prior to validation. Reasoning as an optional pre validation step is beneficial for RDF validation as (1) it may resolve constraint violations and (2) it may cause useful constraint violations. 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. Consider the following DL knowledge base $\mathcal{K}$[21]:

$$
\begin{aligned}
\mathcal{K} = \{\, & \texttt{LogicalDataSet} \sqsubseteq \texttt{dcat:DataSet}, \\
& \texttt{LogicalDataSet} \sqsubseteq \forall \texttt{ aggregation.qb:DataSet}, \\
& \texttt{LogicalDataSet(logical-data-set)}, \\
& \texttt{aggregation(logical-data-set, aggregated-data-set)} \,\}
\end{aligned}
$$

As we know that only person-level data sets (*disco:LogicalDataSet*) can derive (*disco:aggregation*) aggregated data sets (*qb:DataSet*), *logical-data-set* is a *disco:LogicalDataSet*, and *aggregated-data-set* is derived from *logical-data-set*, we conclude that *aggregated-data-set* must be a *qb:DataSet*. As *aggregated-data-set* is not explicitly defined to be a *qb:DataSet*, however, a constraint violation is raised. If we perform reasoning prior to validation, the constraint violation is resolved, as the implicit triple `qb:DataSet(aggregated-data-set)` is inferred.

Reasoning may also cause constraint violations which are needed to enhance data quality. With *subsumption* (*R-100*), one can state that *disco:LogicalDataSet* is a sub-class of *dcat:DataSet*, i.e., each person-level data set is also a catalog data set. Thus, constraints on catalog data sets are also validated for person-level data sets; e.g., the *existential quantification* below restricting that person-level data sets must have a distribution:

[21] A knowledge base is a collection of formal statements which corresponds to *facts* or what is known explicitly. For simplicity reasons, we only write namespace prefixes in DL statements to avoid ambiguities.

$$\mathcal{K} = \{ \texttt{DataSet} \equiv \exists \texttt{ distribution.Distribution}, \\ \texttt{Variable} \equiv \exists \texttt{ concept.Concept} \}$$

We extend $\mathcal{K}$ by *existential quantifications* (*R-86*) enforcing 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 ($\mathcal{SL}_0$). The variable *Education at pre-school* is associated with the theoretical concept *Child Care*. The default severity level of the constraint is weak, as in most cases research can be continued without having information about the theoretical concept of a variable.

$$\mathcal{K} = \{ \texttt{fundedBy} \sqsubseteq \texttt{contributor} \}$$

By stating that *disco:fundedBy* is a sub property of *dcterms:contributor*, the *sub property* (*R-54, R-64*) above assures that if a series is funded by an organization, then the organization must also contribute to the series. In case the *sub-property* is applied without reasoning and $\mathcal{K}$ contains the triple `disco:fundedBy (EU-SILC, organization)`, a constraint violation is thrown if $\mathcal{K}$ does not explicitly include the triple `dcterms:contributor (EU-SILC, organization)`. If the *sub property* is applied with reasoning, on the other side, the latter triple is derived which resolves the constraint violation.

5.2 Basic Constraint Types without Reasoning

$\overline{\mathcal{C}_B^{\mathcal{R}}}$ denotes the set of constraint types for which reasoning cannot be done or does not improve the result in any obvious sense. The constraint type *vocabulary* guarantees that users do not invent new or use deprecated terms of vocabularies. *Value is valid for datatype* (*R-223*) constraints serve to make sure that all literal values are valid with regard to their datatypes - as stated in the vocabularies. Thus, it is checked that all date values (e.g., *dcterms:date*, *disco:startDate*, *disco:endDate*) are actually of the datatype *xsd:date* and that *xsd:nonNegativeInteger* values (e.g. *disco:frequency*) are not negative. 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*).

$$\mathcal{K} = \{ \texttt{isStructuredBy} \sqsubseteq \neg \texttt{ column}, \\ \texttt{TableDescription} \sqcap \texttt{ColumnDescription} \sqsubseteq \bot, \\ \texttt{CategoryStatistics} \equiv \\ \forall \texttt{ computationBase.}\{\texttt{valid,invalid}\} \sqcap \texttt{langString} \}$$

All properties, not having the same domain and range types, are defined to be pairwise disjoint (*R-9: disjoint properties*), i.e., no individual x can be connected to an individual/literal y by disjoint properties (e.g., *phdd:isStructuredBy* and *phdd:column*). All *PHDD* classes (e.g., *phdd:TableDescription*, *phdd:ColumnDescription*) are pairwise disjoint (*R-7: disjoint classes*), i.e., individuals cannot be instances of multiple disjoint classes. 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 values *valid* and *invalid* of the datatype *rdf:langString*. Validation should *exploit sub-super relations* in vocabularies (*R-224*). If *dcterms:coverage* and one of its sub-properties (*dcterms:spatial*, *dcterms:temporal*) are given, it is checked that *dcterms:coverage* is not redundant with its sub-properties which may indicate when the data is verbose/redundant or expressed at a too general level.

6 Simple Constraint Types

$\mathcal{CT}_S$ is the set of constraint types whose constraints can be easily defined without much effort in addition to $\mathcal{CT}_B$ constraints. 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 ($\mathcal{SL}_1$). (2) Language tags must conform to language standards ($\mathcal{SL}_2$). (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 ($\mathcal{SL}_0$) [9].

Default values (*R-31, R-38*) for objects/literals of given properties are inferred automatically when 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*) ($\mathcal{SL}_0$). Many properties are not necessarily required but *recommended* within a particular context (*R-72*). The property *skos:notation*, e.g., is not mandatory for *disco:Variable*s, but recommended to represent variable names ($\mathcal{SL}_0$). Percentage values are only valid when they are within the literal range of 0 and 100 (*R-45: literal ranges*; $\mathcal{SL}_2$) which is checked for *disco:percentage* standing for the number of cases of a given code in relation to the total number of cases for a particular variable.

$$\mathcal{K} = \{\ \texttt{(funct identifier}^{-}\texttt{), identifier keyfor Resource}\ \}$$

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 *Disco*, resources are uniquely identified by the property *adms:identifier*, which is therefore inverse-functional, 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 ($\mathcal{SL}_2$). 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 [10]. Thus and as there are different sorts of key, and as keys can lead to undecidability, *DL* is extended with the construct *keyfor* [8] which is implemented by the *OWL 2 hasKey* construct.

7 Complex Constraint Types

$\mathcal{CT}_C$ denotes the set of constraint types for which the definition of constraints is rather complex and cannot be derived from vocabulary definitions. *Data model consistency* constraints ensure the integrity of the data according to the intended semantics of vocabularies. Every *qb:Observation*, e.g., must have a value for each dimension declared in its *qb:DataStructureDefinition* ($\mathcal{SL}_2$) and no two *qb:Observations* in the same *qb:DataSet* can have the same value for all dimensions ($\mathcal{SL}_1$). If a *qb:DataSet* D has a *qb:Slice* S, and S has an *qb:Observation* O, then the *qb:DataSet* corresponding to O must be D ($\mathcal{SL}_1$). *Mathematical Operations* (*R-41, R-42*; e.g. date calculations and statistical computations like average, mean, and sum) are performed to ensure the integrity of data models. The sum of percentage values of all variable codes, e.g., must exactly be 100 ($\mathcal{SL}_2$) and the minimum absolute frequency of all variable codes do not have to be greater than the maximum ($\mathcal{SL}_2$).

In many cases, resources must be *members of controlled vocabularies* (*R-32*). If a dimension property, e.g., has a *qb:codeList*, then the value of the dimension property on every *qb:Observation* must be in the code list ($\mathcal{SL}_2$). Summary statistics types like minimum, maximum, and arithmetic mean are maintained within a controlled vocabulary. Thus, summary statistics can only have *disco:summaryStatisticType* relationships to *skos:Concept*s which must be members of the controlled vocabulary *ddicv:SummaryStatisticType*, a *skos:ConceptScheme* ($\mathcal{SL}_2$). 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:OrderedCollection*), the variable's code list ($\mathcal{SL}_0$).

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 over a series/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 ($\mathcal{SL}_1$). In case a variable is represented in form of a code list, codes may be associated with categories, i.e., human-readable labels ($\mathcal{SL}_0$). The variable *Education at pre-school*, e.g., is represented as ordered code list without any categories. If a *skos:Concept* represents a code (having *skos:notation* and *skos:prefLabel* properties), then the property *disco:isValid* has to be stated indicating if the code stands for valid (*true*) or missing (*false*) cases ($\mathcal{SL}_2$). *Context-specific exclusive or of property groups* (*R-11*) constraints restrict individuals of given classes to have properties defined within exactly one of multiple property groups. *skos:Concept*s can have either *skos:definition* (when interpreted as theoretical concepts) or *skos:notation* and *skos:prefLabel* properties (when interpreted as codes/categories), but not both ($\mathcal{SL}_2$).

8 Implementation

SPARQL is generally seen as the method of choice to validate RDF data according to certain constraints. We use *SPIN*, a SPARQL-based way to formulate and check constraints, as basis to develop a validation environment (available at http://purl.org/net/rdfval-demo)[22] to validate RDF data according to constraints expressed my arbitrary constraint languages like Shape Expressions, Resource Shapes, and the Web Ontology Language[23] [2]. The *RDF Validator* also validates RDF data to ensure correct syntax, semantics, and integrity of diverse vocabularies such as *Disco*, *QB*, *SKOS*, and *PHDD*. Although accessible within our validation tool, we provide all implemented constraints[24] in form of SPARQL CONSTRUCT queries. For the subsequent evaluation, we implemented 213 constraints on *Disco*, *QB*, *SKOS*, and *PHDD* data sets. The SPIN engine checks for each resource if it satisfies all constraints, which are associated with its assigned classes, and generates a result RDF graph containing information about all constraint violations. There is one SPIN construct template for each constraint type and vocabulary-specific constraint[25]. A SPIN construct template contains a SPARQL CONSTRUCT query which generates constraint violation triples indicating the subject and the properties causing constraint violations, and the reason why constraint violations have been raised. A SPIN construct template creates constraint violation triples if all triple patterns within the SPARQL WHERE clause match. *Missy*[26] provides comprehensive Linked Data services like diverse RDF exports of person-level metadata conforming to the *Disco* vocabulary in form of multiple concrete syntaxes.

9 Evaluation

9.1 Evaluation Setup

1. First, we assigned each constraint type to exactly one of the disjoint sets of constraint types in order to get an overview how many constraint types in relation to the total amount of constraint types are extractable from vocabularies ($\mathcal{CT}_B$), are easily definable ($\mathcal{CT}_S$), and are rather difficult to specify ($\mathcal{CT}_C$).

[22] Source code downloadable at: https://github.com/boschthomas/rdf-validator
[23] SPIN mappings available at: https://github.com/boschthomas/rdf-validation/tree/master/SPIN
[24] https://github.com/boschthomas/rdf-validation/tree/master/constraints
[25] For a comprehensive description of the *RDF Validator*, we refer to [2]
[26] http://www.gesis.org/missy/eu/missy-home

2. Second, several SBE domain experts of the vocabularies *Disco*, *QB*, *SKOS*, *XKOS*, and *PHDD* evaluated the correctness (i.e., the gold standard) of all $\mathcal{CT}_C$ and $\mathcal{CT}_T$ constraints and therefore the generic applicability of the developed classification system of constraint types and constraints.
3. Third, we exhaustively evaluated the metadata quality of large real world aggregated (*QB*), person-level (*Disco*), and thesauri (*SKOS*) data sets by means of both $\mathcal{C}_C$ and $\mathcal{C}_T$ constraints of the majority of the constraint types.

We validated 9,990 / 3,775,983,610 (*QB*), 4,178 / 477,737,281 (*SKOS*), and 1,526 / 9,673,055 (*Disco*) data sets / triples using the *RDF Validator* in batch mode. That are more than 4.2 billion triples and 15 thousand data sets. We validated, i.a., (1) *QB* data sets published by the *Australian Bureau of Statistics (ABS)*, the *European Central Bank (ECB)*, and the *Organisation for Economic Co-operation and Development (OECD)*, (2) *SKOS* thesauri like the *AGROVOC Multilingual agricultural thesaurus*, the *STW Thesaurus for Economics*, and the *Thesaurus for the Social Sciences (TheSoz)*, and (3) *Disco* data sets provided by the *Microdata Information System (Missy)*, the *DwB Discovery Portal*, the *Danish Data Archive (DDA)*, and the *Swedish National Data Service (SND)*. We recently published a technical report[27] (serving as second appendix of this paper) in which we describe the comprehensive evaluation in detail [4]. As we evaluated nearly 10 thousand *QB* data sets, we published the evaluation results for each data set in form of one document per SPARQL endpoint[28].

9.2 Evaluation Results and Discussion

The majority ($48 \equiv 58.5\%$) of the overall 82 $\mathcal{CT}$ constraint types are $\mathcal{CT}_B$ whose constraints can therefore be derived from vocabularies without any effort. Among $\mathcal{CT}_B$, two-thirds ($34 \equiv 70.8\%$) are $\mathcal{C}_B^{\mathcal{R}}$, i.e., constraint types for which reasoning may be performed prior to validation, and one third ($14 \equiv 29.2\%$) are $\overline{\mathcal{C}_B^{\mathcal{R}}}$, i.e., constraint types for which reasoning does not make any sense. A quarter ($20 \equiv 24.4\%$) of all constraint types are $\mathcal{CT}_S$ and a sixth ($14 \equiv 17.1\%$) are $\mathcal{CT}_C$.

Criteria	***Disco***	***QB***	***SKOS***	**Total**
$\mathcal{CT}$	52	20	15	53
$\mathcal{CT}_C$	9 (17.3%)	2 (10%)	2 (13.3%)	9 (17%)
$\mathcal{CT}_S$	16 (30.8%)	3 (15%)	4 (26,7%)	16 (30.2%)
$\mathcal{CT}_B$	27 (**51.9%**)	15 (**75%**)	9 (**60%**)	28 (**52.8%**)
$\mathcal{C}_B^{\mathcal{R}}$	18 (34.6%)	9 (45%)	4 (26.7%)	19 (35.9%)
$\overline{\mathcal{C}_B^{\mathcal{R}}}$	9 (17.3%)	6 (30%)	5 (33.3%)	9 (17%)

Table 1: Evaluation - Constraint Types

[27] Available at: http://arxiv.org/abs/1504.04478
[28] Available at: https://github.com/boschthomas/rdf-validation/tree/master/evaluation/data-sets/data-cube

We defined constraints of 53 distinct $\mathcal{CT}$ constraint types (see table 1). More than the half of them are $\mathcal{CT}_B$, more than a third $\mathcal{C}^{\mathcal{R}}_{B}$, nearly a third $\mathcal{CT}_S$, and only a sixth $\mathcal{CT}_C$ constraint types. For *Disco*, *QB*, and *SKOS*, more than 50% of the instantiated constraint types are $\mathcal{CT}_B$ constraint types (for *QB* even three quarters). *Existential quantifications* (*R-86*, 32.4%, *Disco*), *data model consistency* (31.4%, *QB*), and *structure* (28.6%, *SKOS*) are the constraint types the most constraints are instantiated from.

Criteria	***Disco***	***QB***	***SKOS***	**Total**
C	143 (67.1%)	35 (16.4%)	35 (16.4%)	213
$C\ (\mathcal{CT}_C)$	37 (25.9%)	13 (37.1%)	13 (**37.1%**)	63 (29.6%)
$C\ (\mathcal{CT}_S)$	28 (19.6%)	3 (8.6%)	12 (**34.3%**)	43 (20.2%)
$C\ (\mathcal{CT}_B)$	78 (**54.6%**)	19 (**54.3%**)	10 (**28.6%**)	107 (**50.2%**)
$C\ (\mathcal{C}^{\mathcal{R}}_{B})$	67 (46.9%)	13 (37.1%)	4 (11.4%)	84 (39.4%)
$C\ (\overline{\mathcal{C}^{\mathcal{R}}_{B}})$	11 (7.7%)	6 (17.1%)	6 (17.1%)	23 (10.8%)
$C\ (\mathcal{SL}_0)$	75 (**52.5%**)	4 (11.4%)	21 (**60%**)	100 (**46.9%**)
$C\ (\mathcal{SL}_1)$	10 (7%)	3 (8.6%)	5 (14.3%)	18 (8.5%)
$C\ (\mathcal{SL}_2)$	58 (40.6%)	28 (**80%**)	9 (25.7%)	95 (**44.6%**)

Table 2: Evaluation - Constraints

Half of the overall 213 $\mathcal{CT}$ constraints are $\mathcal{CT}_B$ constraints (almost 40% $\mathcal{C}^{\mathcal{R}}_{B}$) (see table 2). This is not surprising as more than the half of the instantiated constraints types are $\mathcal{CT}_B$ constraint types. More than 50% of the *Disco* and *QB* constraints are $\mathcal{CT}_B$ constraints. The *SKOS* constraints, on the other side, are equally assigned to the three disjoint constraint type sets. *SBE* domain experts associated almost the half of the constraints with the severity levels $\mathcal{SL}_0$ (*informational*) (47%) and $\mathcal{SL}_2$ (*error*) (45%); only 8.5% are $\mathcal{SL}_1$ (*warning*) constraints. The violation of 80% of the *QB* constraints is very serious ($\mathcal{SL}_2$). More than 50% of the *Disco* and *SKOS* constraints, however, are assigned to the weakest default severity level ($\mathcal{SL}_0$).

Criteria	***Disco***	***QB***	***SKOS***	**Total**
CV	3,575,002 (6.5%)	45,635,861 (83.4%)	5,540,988 (10.1%)	54,751,851
$CV\ (\mathcal{CT}_C)$	652,780 (18.3%)	45,634,084 (**100%**)	1,185,982 (21.4%)	47,472,846 (**86.7%**)
$CV\ (\mathcal{CT}_S)$	560,857 (15.7%)	0 (0%)	4,355,006 (**78.6%**)	4,915,863 (9%)
$CV\ (\mathcal{CT}_B)$	2,361,365 (**66.1%**)	1,777 (0%)	0 (0%)	2,363,142 (4.3%)
$CV\ (\mathcal{C}_B^{\mathcal{R}})$	2,333,365 (65.3%)	1,777 (0%)	0 (0%)	2,335,142 (4.3%)
$CV\ (\overline{\mathcal{C}_B^{\mathcal{R}}})$	28,000 (0.8%)	0 (0%)	0 (0%)	28,000 (0.1%)
$CV\ (\mathcal{SL}_0)$	1,880,777 (**52.6%**)	0 (0%)	2,281,740 (41.2%)	4,162,517 (7.6%)
$CV\ (\mathcal{SL}_1)$	1,051,757 (29.4%)	45,520,613 (**99,8%**)	3,259,248 (**58.8%**)	49,831,618 (**91%**)
$CV\ (\mathcal{SL}_2)$	642,468 (18%)	115,248 (0.3%)	0 (0%)	757,716 (1.4%)

Table 3: Evaluation - Constraint Violations

More than 80% of the overall approx. 55 million constraint violations are caused by *QB* constraints (see table 3). The majority (86.7%) of the constraint violations are caused by $\mathcal{CT}_C$ constraints and only 9% by $\mathcal{CT}_S$ constraints. The fact that only 4% of all constraint violations result from $\mathcal{CT}_B$ constraints, even though, more than 50% of all constraints are $\mathcal{CT}_B$ constraints, is an indicator that data producers achieve good data quality with respect to the constraints extracted by vocabularies. Two-thirds of the constraint violations, thrown for *Disco* data sets, result from $\mathcal{CT}_B$ constraints, almost only $\mathcal{CT}_C$ constraints raised constraint violations for *QB* data sets, and nearly 80% of the *SKOS* constraint violations are caused by $\mathcal{CT}_S$ constraints. Although, only 8.5% of all constraints are $\mathcal{SL}_1$ constraints, 9 0f 10 constraint violations are caused by constraints associated with the default severity level $\mathcal{SL}_1$. More than 50% of all constraint violations thrown for *Disco* data sets are not seen to be that significant ($\mathcal{SL}_0$). Almost all constraint violations raised by *QB* constraints and 59% of the constraint violations caused by *SKOS* constraints are classified as warnings ($\mathcal{SL}_1$). As this evaluation serves as indication for *good* or *bad* (meta)data quality, we conclude that the metadata quality of all evaluated *Disco* data sets is *good* and that the metadata quality of all evaluated *QB* and *SKOS* data sets is *satisfactory.*

10 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 XSDs[29]. As certain constraints cannot be formulated and validated by XSDs, so-called secondary-level validation tools like *Schematron*[30] have been introduced to overcome the limitations of XML validation. *Schematron* generates validation rules and validates XML documents according to them.

[29] http://www.ddialliance.org/Specification/

[30] https://msdn.microsoft.com/en-us/library/aa468554.aspx

With RDF validation, one can overcome drawbacks when validating XML documents[31]. It cannot be validated, e.g., if each code of a variable's code list is associated with a category (*R-86*). Additionally, it cannot be validated that if an element has a specific value, then certain child elements must be present (*R-71*). A comprehensive comparison of XML and RDF validation, however, is not within the scope of this paper.

A well-formed *RDF Data Cube* is an a RDF graph describing one or more instances of *qb:DataSet* for which each of the 22 integrity constraints[32], defined within the *QB* specification, passes. Each integrity constraint is expressed as narrative prose and, where possible, 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 [7]. Mader, Haslhofer, and Isaac 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 [9].

11 Conclusion and Future Work

Thomas: ToDO

[33]

We implemented a validation environment (available at http://purl.org/net/rdfval-demo) to validate RDF data according to constraints expressed my arbitrary constraint languages and to ensure correct syntax and semantics of diverse vocabularies such as *Disco*, *QB*, *SKOS*, and *PHDD* (section 8). We exhaustively evaluated the metadata quality of large real world aggregated (*QB*), person-level (*Disco*), and thesauri (*SKOS*) data sets (more than 4.2 billion triples and 15 thousand data sets) by means of 213 constraints of the majority of the constraint types[34] (section 9).

References

1. Thomas Bosch and Kai Eckert. Requirements on rdf constraint formulation and validation. *Proceedings of the DCMI International Conference on Dublin Core and Metadata Applications (DC 2014)*, 2014.
2. Thomas Bosch and Kai Eckert. Towards description set profiles for rdf using sparql as intermediate language. *Proceedings of the DCMI International Conference on Dublin Core and Metadata Applications (DC 2014)*, 2014.

[31] http://www.xmlmind.com/xmleditor/_distrib/doc/xmltool/xsd_structure_limitations.html

[32] http://www.w3.org/TR/vocab-data-cube/#wf

[33] The first appendix of this paper describing each constraint in detail is available at: http://arxiv.org/abs/1504.04479 [5]

[34] The second appendix of this paper describing the evaluation in detail is available at: http://arxiv.org/abs/1504.04478 [4].

3. Thomas Bosch, Andreas Nolle, Erman Acar, and Kai Eckert. Rdf validation requirements - evaluation and logical underpinning. 2015.
4. Thomas Bosch, Benjamin Zapilko, Joachim Wackerow, and Kai Eckert. An evaluation of metadata and data quality on person-level, aggregated, thesauri, statistical classifications, and rectangular data sets. 2015.
5. Thomas Bosch, Benjamin Zapilko, Joachim Wackerow, and Kai Eckert. Rdf constraints to validate rectangular data and metadata on person-level data, aggregated data, thesauri, and statistical classifications. 2015.
6. Richard Cyganiak, Simon Field, Arofan Gregory, Wolfgang Halb, and Jeni Tennison. Semantic statistics: Bringing together sdmx and scovo. In Christian Bizer, Tom Heath, Tim Berners-Lee, and Michael Hausenblas, editors, *Proceedings of the WWW 2010 Workshop on Linked Data on the Web*, volume 628 of *CEUR Workshop Proceedings*, 2010.
7. Richard Cyganiak and Dave Reynolds. The rdf data cube vocabulary. W3C recommendation, W3C, January 2014.
8. Carsten Lutz, Carlos Areces, Ian Horrocks, and Ulrike Sattler. Keys, nominals, and concrete domains. *Journal of Artificial Intelligence Research*, 23(1):667–726, June 2005.
9. Christian Mader, Bernhard Haslhofer, and Antoine Isaac. Finding quality issues in skos vocabularies. In *Proceedings of the Second International Conference on Theory and Practice of Digital Libraries*, TPDL'12, pages 222–233, Berlin, Heidelberg, 2012. Springer-Verlag.
10. Michael Schneider. OWL 2 Web Ontology Language RDF-Based Semantics. W3C recommendation, W3C, October 2009.
11. Mary Vardigan, Pascal Heus, and Wendy Thomas. Data documentation initiative: Towards a standard for the social sciences. *International Journal of Digital Curation*, 3(1):107–113, 2008.