Validation Framework for RDF-based Constraint Languages

PHD THESIS APPENDIX

Department of Economics and Management

Karlsruhe Institute of Technology (KIT)

M.Sc. Thomas Hartmann

Abstract

This paper serves as appendix for the PhD thesis entitled *Validation Framework for RDF-based Constraint Languages*, submitted to the Department of Economics and Management at the Karlsruhe Institute of Technology (KIT).

The formulation of constraints and the validation of RDF data against these constraints is a common requirement and a much sought-after feature, particularly as this is taken for granted in the XML world. Recently, RDF validation as a research field gained speed due to shared needs of data practitioners from a variety of domains. For constraint formulation and RDF data validation, several languages exist or are currently developed. Yet, there is no clear favorite and none of the languages is able to meet all requirements raised by data professionals. Therefore, further research on RDF validation and the development of constraint languages is needed.

There are different types of research data and related metadata. Because of the lack of suitable RDF vocabularies, however, just a few of them can be expressed in RDF. We have developed three missing vocabularies to represent all types of research data and its metadata in RDF and to validate RDF data according to constraints extractable from these vocabularies.

Data providers of many domains still represent their data in XML, but expect to increase the quality of their data by using common RDF validation tools. In order to be able to directly validate XML against semantically rich OWL axioms when using them in terms of constraints and extracting them from XML Schemas adequately representing particular domains, we propose on formal logics and the XML Schema meta-model based automatic transformations of arbitrary XML Schemas and corresponding XML documents into OWL ontologies and conforming RDF data without any information loss and without having any additional manual effort defining constraints.

We have published a set of constraint types that are required by diverse stakeholders for data applications and which form the basis of this thesis. Each constraint type, from which concrete constraints are instantiated to be checked on the data, corresponds to one of the requirements contained in a community-driven database of requirements to formulate constraints and validate RDF data. We have initiated this database to collaboratively collect case studies provided by various data institutions, use cases, requirements, and solutions in a comprehensive and structured

way. We use this collection of constraint types to gain a better understanding of the expressiveness of existing and currently evolved solutions, identify gaps that still need to be filled, recommend possible solutions for their elimination, and give directions for the further development of constraint languages.

SPARQL is generally seen as the method of choice to validate RDF data, although it is not ideal for constraint formulation. In contrast, high-level constraint languages are comparatively easy to understand and allow to formulate constraints in a more concise way, but either lack an implementation to actually validate RDF data against constraints expressed in these languages or are based on different implementations.

We introduce a validation framework that enables to consistently execute RDF-based constraint languages on RDF data and to formulate constraints of any type in a way that mappings from high-level constraint languages to an intermediate generic representation can be created straight-forwardly. The framework reduces the representation of constraints to the absolute minimum, is based on formal logics, and consists of a very simple conceptual model using a small lightweight vocabulary. We demonstrate that using another layer on top of SPARQL ensures consistency regarding validation results and enables constraint transformations for each constraint type across RDF-based constraint languages.

The constraint types form the basis to investigate the role that reasoning plays in practical data validation, when reasoning is beneficial for RDF validation, and how to overcome the major shortcomings when validating RDF data by performing reasoning prior to validation. For each constraint type, we investigate (1) if reasoning may be executed to enhance data quality, (2) how efficient in terms of runtime validation is performed with and without reasoning, and (3) if validation results differ when different semantics is assumed.

We evaluate the usability of constraint types for assessing RDF data quality by (1) collecting and classifying constraints on vocabularies, either from the vocabularies themselves or from domain experts, and (2) validating 15,694 data sets (4.26 billion triples) of research data according to these constraints. Based on the large-scale evaluation, we formulate several findings to direct the further development of constraint languages.

Contents

A

Types of Constraints on RDF Data

We initiated a community-driven database of requirements to formulate constraints and validate RDF data against these constraints. The intention of this database is to collaboratively collect case studies provided by various data institutions, use cases, requirements, and solutions in a comprehensive and structured way. The database is publicly available at http://purl.org/net/rdf-validation, continuously extended, and open for further contributions.

Based on our work in the *DCMI RDF Application Profiles Task Group*[1] and the *W3C RDF Data Shapes Working Group*[2] and the requirements jointly identified within these working groups, we have published by today 81 types of constraints that are required by various stakeholders for data applications; each constraint type, from which concrete constraints are instantiated to be checked on the data, corresponds to a specific requirement in the database.

For each constraint type, we give a formal definition and a detailed explanation in form of intuitive example constraints. In a technical report, we provide additional examples for each constraint type represented in different constraint languages [18].

We instantiate each constraint type at least once and thereby demonstrate how to generically express constraints (1) in *Description Logics (DL)*,[3] on condition that their constraint type is expressible in DL, and (2) using the *RDF Constraints Vocabulary (RDF-CV)*,[4] a small lightweight vocabulary introduced within the developed validation framework.

For each constraint type, we provide templates showing how arbitrary constraints of a given constraint type are representable using the RDF-CV in a generic way. In case a particular constraint must hold for all individuals of

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

[2] http://www.w3.org/2014/rds/charter

[3] DL statements are contained in a *DL knowledge base* $\mathcal{K}$ which is a collection of formal statements corresponding to *facts* or what is known explicitly.

[4] Formal specification, HTML documentation, and UML class diagram online available at: https://github.com/github-thomas-hartmann/phd-thesis

the graph to be validated, the context class of the generic representation is set to the *DL top concept* $\top$, a special concept with every individual as an instance.

A.1 Functional Properties

Constraints of the constraint type *functional properties (R-57/65)* state that the object or data properties p_i $(1 \leqslant i \leqslant n)$ are functional within the context of the class $C_{context}$ - that is, for each individual i_1 of the class $C_{context}$, there can be at most one distinct individual i_2 such that i_1 is connected by p_i to i_2. As the property *title* is functional, a book can have at most one distinct title. The data property *isbn* is functional, since books can only have one ISBN.

$$\mathcal{K} = \{ \texttt{funct (isbn)} \}$$

Table A.1. Generic Representation of the Constraint of the Constraint Type *Functional Properties*

context class	left p. list	right p. list	classes	c. element	c. value
Book	isbn	-	-	functional properties	-

Table A.2. Generic Representation of the Constraint Type *Functional Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p_i>$ $(1 \leqslant i \leqslant n)$	-	-	functional properties	-

A.2 Inverse-Functional Properties

Constraints of the constraint type *inverse-functional properties (R-58)* state that the object properties p_i $(1 \leqslant i \leqslant n)$ are inverse-functional within the context of the class $C_{context}$ - that is, for each individual i_1, there can be at most one individual i_2 such that i_2 is connected by p_i with i_1. In DDI-RDF, resources are uniquely identified by the property *adms:identifier*, which is therefore inverse-functional.

$$\mathcal{K} = \{ \text{funct (identifier}^{-}) \}$$

Table A.3. Generic Representation of the Constraint of the Constraint Type *Inverse-Functional Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	adms:identifier	-	-	inverse-functional properties	-

Table A.4. Generic Representation of the Constraint Type *Inverse-Functional Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p_i>$ $(1 \leqslant i \leqslant n)$	-	-	inverse-functional properties	-

A.3 Primary Key Properties

The *primary key properties (R-226)* constraint type is often useful to declare a given (data) property p as the primary key of a class $C_{context}$, so that a system can enforce uniqueness. Books, e.g., are uniquely identified by their ISBN, i.e., the property *isbn* is inverse functional (`funct` $isbn^-$). The meaning of this constraint is that ISBN identifiers can only have $isbn^-$ relations to at most one distinct book. Keys, however, are even more general, i.e., a generalization of inverse functional properties [32]. A key can be a data, an object property, or a chain of properties. For these generalization purposes, as there are different sorts of keys, and as keys can lead to undecidability, DL is extended with a special construct *keyfor* (`isbn keyfor Book`) [28].

$$\mathcal{K} = \{\ \mathsf{funct}\ (\mathsf{isbn}^-)\ \}$$

Table A.5. Generic Representation of the Constraint of the Constraint Type *Primary Key Properties*

context class	left p. list	right p. list	classes	c. element	c. value
Book	isbn	-	-	primary key	-

Table A.6. Generic Representation of the Constraint Type *Primary Key Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p>$	-	-	primary key	-

A.4 Subsumption

The *subsumption (R-100)* constraint type corresponds to *concept inclusion* in DL terminology. Sub-class axioms are a fundamental type of axioms in OWL 2 and can be used to construct class hierarchies. If C_1 is a sub-class of C_2, i.e., C_1 is more specific than C_2, then each instance of C_1 must also be an instance of C_2. Each book, e.g., must also be of the type publication.

$$\mathcal{K} = \{\ \texttt{Book} \sqsubseteq \texttt{Publication}\ \}$$

Table A.7. Generic Representation of the Constraint of the Constraint Type *Subsumption*

context class	left p. list	right p. list	classes	c. element	c. value
Book	-	-	Publication	sub-class	-

Table A.8. Generic Representation of the Constraint Type *Subsumption*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_1>$	-	-	$<C_2>$	sub-class	-

A.5 Sub-Properties

Sub-properties (R-54/64) (*role inclusion* in DL) are analogous to subsumption, i.e., sub-class relationships. With sub-properties, one can state that the property p_1 is a sub-property of the property p_2 - that is, if an individual i_1 is connected by p_1 to an individual i_2 or a literal l, then i_1 is also connected by p_2 to i_2/l. If a journal volume has an *editor* relationship to a person, e.g., then the journal volume must also have a *creator* link to the same person, i.e., *editor* is a sub-property of *creator*. If we validate against this sub-properties constraint and the data contains the triple `editor (A+Journal-Volume, A+Editor),` then the triple `creator (A+Journal-Volume, A+Editor)` has to be stated explicitly to prevent the constraint to be violated. In contrast, if the second triple is not present in the data, a violation occurs.

$$\mathcal{K} = \{\ \texttt{editor} \sqsubseteq \texttt{creator}\ \}$$

Table A.9. Generic Representation of the Constraint of the Constraint Type *Sub-Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	editor	creator	-	sub-property	-

Table A.10. Generic Representation of the Constraint Type *Sub-Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	$<p_1>$	$<p_2>$	-	sub-property	-

A.6 Object Property Paths

Object property paths (R-55) (or *object property chains* and in DL terminology *complex role inclusion* or *role composition*) is the more complex form of the *sub-properties* constraint type. With object property paths, one can state that, if an individual i_1 is connected by a sequence of object properties p_1, ..., p_n with an individual i_2, then i_1 is also connected with i_2 by the object property p. As *Stephen-Hawking* is the author of the book *A-Brief-History-Of-Time* whose genre is *Popular-Science*, the object property path $\texttt{authorOf} \circ \texttt{genre} \sqsubseteq \texttt{authorOfGenre}$ infers that *Stephen-Hawking* is an author of the genre *Popular-Science*. In case the last triple is not present in the data, the object property paths constraint is violated.

$$\mathcal{K} = \{\ \texttt{authorOf} \circ \texttt{genre} \sqsubseteq \texttt{authorOfGenre}\ \}$$

Table A.11. Generic Representation of the Constraint of the Constraint Type *Object Property Paths*

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

Table A.12. Generic Representation of the Constraint Type *Object Property Paths*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	$<p_i>$ $(1 \leqslant i \leqslant n)$	$<p>$	-	object property path	-

A.7 Allowed Values

It is a common requirement to narrow down the value space of properties by an exhaustive enumeration of valid values - both literals or resources (*R-30/37: allowed values*). This is often rendered in drop down boxes or radio buttons in user interfaces. Allowed values for property values may be IRIs, IRIs matching specific patterns, IRIs matching one of multiple patterns, any literals, literals of a list of allowed literals, or typed literals of a certain datatype. A constraint of this type ensures that all instances of a given class $C_{context}$ can only have relations via a specific object or data property p to individuals i_i or literals l_i $(1 \leqslant i \leqslant n)$ of a set of allowed individuals/literals. Consider the following example of a constraint of this type which states that books on the topic computer science can only have *Computer-Science* and *Informatics* as allowed subjects.

$$\mathcal{K} = \{\ \texttt{Computer-Science-Book} \sqsubseteq \forall\ \texttt{subject.\{Computer-Science\}} \sqcup \texttt{\{Informatics\}}\ \}$$

Table A.13. Generic Representation of the Constraint of the Constraint Type *Allowed Values*

context class	left p. list	right p. list	classes	c. element	c. value
Computer-Science-Book	subject	-	Computer-Science, Informatics	allowed values	-

Table A.14. Generic Representation of the Constraint Type *Allowed Values*

context class	left p. list	right p. list	classes		c. element	c. value
$<C_{context}>$	$<p>$	-	$<i_i>$	$<l_i>$ $(1 \leqslant i \leqslant n)$	allowed values	-

A.8 Not Allowed Values

A constraint of the constraint type *not allowed values (R-33/200)* ensures that all instances of a given class $C_{context}$ cannot have relations via a specific object or data property p to individuals i_i or literals l_i $(1 \leqslant i \leqslant n)$ of a set of not allowed individuals/literals. Consider the following example of a

constraint of this type which states that books on the topic computer science cannot have *Economic-Sciences* or *Business-Sciences* as subjects.

$$\mathcal{K} = \{\ \texttt{Computer-Science-Book} \equiv \neg\ \exists\ \texttt{subject.\{Economic-Sciences\}} \sqcup \texttt{\{Business-Sciences\}}\ \}$$

Table A.15. Generic Representation of the Constraint of the Constraint Type *Not Allowed Values*

context class	left p. list	right p. list	classes	c. element	c. value
Computer-Science-Book	subject	-	Economic-Sciences, Business-Sciences	not allowed values	-

Table A.16. Generic Representation of the Constraint Type *Not Allowed Values*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p>$	-	$<i_i>$ | $<l_i>$ $(1 \leqslant i \leqslant n)$	not allowed values	-

A.9 Class Equivalence

Constraints of the constraint type *class equivalence (R-3)*, or *concept equivalence* in DL terminology, assert that two classes have the same instances. While synonyms are an obvious example of equivalent classes, in practice, class equivalence is more often used to give a name to complex expressions [27]. Class equivalence is indeed subsumption from left to right and from right to left ($C_1 \sqsubseteq C_2$ and $C_2 \sqsubseteq C_1$ implies $C_1 \equiv C_2$). Constraints of this constraint type state that all of the classes C_i, $1 \leqslant$ i $\leqslant$ n, are semantically equivalent to each other. This constraint type allows one to use each C_i as a synonym for each C_j - that is, any class C_i can be replaced with C_j without affecting the meaning of a constraint. Each person is a human and each human is a person, is an example of a constraint of the type *class equivalence*.

$$\mathcal{K} = \{\ \texttt{Person} \equiv \texttt{Human}\ \}$$

Table A.17. Generic Representation of the Constraint of the Constraint Type *Class Equivalence*

context class	left p. list	right p. list	classes	c. element	c. value
Person	-	-	Human	equivalent classes	-

Table A.18. Generic Representation of the Constraint Type *Class Equivalence*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_1>$	-	-	$<C_2>$	equivalent classes	-

A.10 Equivalent Properties

A constraint of the constraint type *equivalent properties (R-4/5)* states that all of the object or data properties p_i $(1 \leqslant i \leqslant n)$ are semantically equivalent to each other. This constraint type allows one to use each p_i as a synonym for each p_j — that is, in any constraint, p_i can be replaced with p_j without affecting the meaning of the constraint. The property *author*, e.g., should be handled as a synonym of the property *hasAuthor*.

$$\mathcal{K} = \{\ \texttt{author} \equiv \texttt{hasAuthor}\ \}$$

Table A.19. Generic Representation of the Constraint of the Constraint Type *Equivalent Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	author, hasAuthor	-	-	equivalent properties	-

Table A.20. Generic Representation of the Constraint Type *Equivalent Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	$<p_i>$ $(1 \leqslant i \leqslant n)$	-	-	equivalent properties	-

A.11 Literal Value Comparison

Constraints of the constraint type *literal value comparison (R-43)* ensure that, depending on the datatype DT of data properties p_1 and p_2, two different literal values of the data properties p_1 and p_2 have a specific ordering with respect to an operator like $>$, $\geqslant$, $<$, $\leqslant$, $=$, and $\neq$. Constraints of this constraint type are checked for each instance of a given class $C_{context}$ which may be $\top$ in case constraints should be checked for each individual of the input graph. It has to be guaranteed, e.g., that birth dates of persons are before $(<)$ death dates. If the birth and the death date of *Albert-Einstein* are interchanged `(birthDate(Albert-Einstein, "1955-04-18"), deathDate(Albert-Einstein, "1879-03-14"))`, a violation is thrown.

Table A.21. Generic Representation of the Constraint of the Constraint Type *Literal Value Comparison*

context class	left p. list	right p. list	classes	c. element	c. value
Person	birthDate	deathDate	xsd:date	is less than	-

Table A.22. Generic Representation of the Constraint Type *Literal Value Comparison*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p_1>$	$<p_2>$	<DT>	<mathematical symbol>	-

A.12 Value is Valid for Datatype

Constraints of the constraint type *value is valid for datatype (R-223)* enable to make sure that for $C_{context}$ individuals each literal of the property p is valid for its datatype DT. It has to be ensured, e.g., that a literal is actually (1) a date, (2) numeric, (3) a string, or (4) an integer which is allowed to be negative or not. By means of this type it can be checked if all literal values of all properties of the datatype xsd:date which are used within the context of DDI-RDF (e.g., disco:startDate, disco:endDate, and dcterms:date) are really of the datatype xsd:date. The subsequent constraint of this type enforces that stated numbers of pages of publications must be integer values which must not be negative.

Table A.23. Generic Representation of the Constraint of the Constraint Type *Value is Valid for Datatype*

context class	left p. list	right p. list	classes	c. element	c. value
Publication	numberPages	-	xsd:nonNegativeInteger	value is valid for datatype	-

Table A.24. Generic Representation of the Constraint Type *Value is Valid for Datatype*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p>$	-	$<DT>$	value is valid for datatype	-

A.13 Property Domains

Constraints of the *property domains* (*R-25*) constraint type restrict the domain of object or data properties. In DL terminology, this constraint type is also called *domain restrictions on roles*. The purpose of this constraint type is to declare that a given object or data property p is associated with a class C, e.g., to populate input forms with appropriate widgets. In object-oriented terms, this is the declaration of a member field, attribute, or association. $\exists p.\top \sqsubseteq C$ is the object property restriction where p is the object property whose domain is restricted to the class C. A constraint of the type property domains states that the domain of the object or data property p is the class C - that is, if an individual i is connected by p with some other individual or some literal, then i is an instance of C. The property domains constraint $\exists$ `author.`$\top \sqsubseteq$ `Publication`, e.g., ensures that only publications can have *author* relationships. The triple `author(Alices-Adventures-In-Wonderland, Lewis-Carroll)` leads to a violation if it is not explicitly stated that *Alices-Adventures-In-Wonderland* is a publication.

$$\mathcal{K} = \{\ \exists\ \texttt{author.}\top \sqsubseteq \texttt{Publication}\ \}$$

Table A.25. Generic Representation of the Constraint of the Constraint Type *Property Domains*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	author	-	Publication	property domain	-

Table A.26. Generic Representation of the Constraint Type *Property Domains*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	$<p>$	-	$<C>$	property domain	-

A.14 Property Ranges

Constraints of the constraint type *property ranges* (*R-35*) restrict the range of object and data properties. In DL terminology, this constraint type is also called *range restrictions on roles*. $\top \sqsubseteq \forall p.C$ is the range restriction on the object property p, restricted by the class C. The property ranges constraint type states that the range of the object or data property p is either (1) the class C - that is, if some individual is connected by p with an individual i, then i is an instance of C, or (2) the datatype DT - that is, if some individual is connected by p with a literal l, then l is in DT. By means of a property ranges constraint it can be restricted that *author* relations can only point to persons. If *Doyle*, the author of the book *Sherlock-Holmes* `(author(Sherlock-Holmes, Doyle))`, e.g., is not explicitly declared to be a person, a constraint violation is caused.

$$\mathcal{K} = \{\ \top \sqsubseteq \forall\ \texttt{author.Person}\ \}$$

Table A.27. Generic Representation of the Constraint of the Constraint Type *Property Ranges*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	author	-	Person	property range	-

Table A.28. Generic Representation of the Constraint Type *Property Ranges*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	$<p>$	-	$<C>$	property range	-

A.15 Class-Specific Property Range

Constraints of the constraint type *class-specific property range (R-29/36)* restrict the range of object and data properties only for individuals of a given class $C_{context}$. Constraints of the class-specific property range constraint type state that for $C_{context}$ individuals the range of the object or data property p

is either (1) the class C - that is, if some $C_{context}$ individual is connected by p with an individual i, then i is an instance of C, or (2) the datatype DT - that is, if some $C_{context}$ individual is connected by p with a literal l, then l is in DT. Constraints of this type must only hold for individuals of a given class $C_{context}$. By means of a class-specific property range constraint it can be restricted that books can only have *author* relations that point to persons. As *Doyle*, the author of the book *Sherlock-Holmes* `(author(Sherlock-Holmes, Doyle), Book(Sherlock-Holmes))`, e.g., is not explicitly declared to be a person, a constraint violation is caused.

$$\mathcal{K} = \{\ \texttt{Book} \sqsubseteq \forall\ \texttt{author.Person}\ \}$$

Table A.29. Generic Representation of the Constraint of the Constraint Type *Class-Specific Property Range*

context class	left p. list	right p. list	classes	c. element	c. value
Book	author	-	Person	property range	-

Table A.30. Generic Representation of the Constraint Type *Class-Specific Property Range*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p>$	-	$<C>$	property range	-

A.16 Data Property Facets

Similar to the facets of XML Schema datatypes, it should be easy to define frequently needed facets for data properties p within the scope of classes $C_{context}$ to drive user interfaces and validate input against simple conditions, including min/max values (similar to *xsd:maxInclusive*, *xsd:maxExclusive*, *xsd:minExclusive*, and *xsd:minInclusive*), pattern matching against regular expressions (*xsd:pattern*), and string length (*xsd:length*, *xsd:minLength*, and *xsd:maxLength*). Additional constraining facets to restrict datatypes of RDF literals are *xsd:enumeration*, *xsd:whiteSpace*, *xsd:total*, *xsd:totalDigits*, and *xsd:fractionDigits*. An example constraint of the constraint type *data property facets (R-46)* is to restrict the abstract of studies to have a minimum length (*xsd:minLength*) of 20 characters in order to be useful for researchers.

Table A.31. Generic Representation of the Constraint of the Constraint Type *Data Property Facets*

context class	left p. list	right p. list	classes	c. element	c. value
Study	abstract	-	-	xsd:minLength	20

Table A.32. Generic Representation of the Constraint Type *Data Property Facets*

context class	left p. list	right p. list	classes	c. element	c. value
<$C_{context}$>	<p>	-	-	<SPARQL function> | <XML Schema constraining facet>	<c. value>

A.17 Literal Ranges

Constraints of the constraint type *literal ranges (R-45)* ensure that for $C_{context}$ individuals the literal values of the data property p of the datatype DT are within the *literal range* of [V_{min},V_{max}]. The latitude of a spatial feature, e.g., must be within the literal range of [-90,90].

Table A.33. Generic Representation of the Constraint of the Constraint Type *Literal Ranges*

context class	left p. list	right p. list	classes	c. element	c. value
Spatial-Feature	latitude	-	xsd:nonNegativeInteger	xsd:minInclusive	-90
Spatial-Feature	latitude	-	xsd:nonNegativeInteger	xsd:maxInclusive	90

Table A.34. Generic Representation of the Constraint Type *Literal Ranges*

context class	left p. list	right p. list	classes	c. element	c. value
<$C_{context}$>	<p>	-	<DT>	<XSD constraining facet on values>	<literal range>

A.18 Negative Literal Ranges

Constraints of the constraint type *negative literal ranges (R-142)* ensure that for individuals of a given class $C_{context}$ the literal values of a particular data property p of the datatype DT are outside a specific *literal range* of [V_{min},V_{max}]. The longitude of a spatial feature, e.g., must not be within [181,360].

Table A.35. Generic Representation of the Constraint of the Constraint Type *Negative Literal Ranges*

context class	left p. list	right p. list	classes	c. element	c. value
Spatial-Feature	longitude	-	xsd:nonNegativeInteger	not xsd:minInclusive	181
Spatial-Feature	longitude	-	xsd:nonNegativeInteger	not xsd:maxInclusive	360

Table A.36. Generic Representation of the Constraint Type *Negative Literal Ranges*

context class	left p. list	right p. list	classes	c. element	c. value
<$C_{context}$>	<p>	-	<DT>	not <XSD constraining facet on values>	<negative literal range>

A.19 IRI Pattern Matching

Constraints of the constraint type *IRI pattern matching (R-21/22/23)* can be applied on subjects, predicates, and objects.

Applied on subjects, constraints of this type enable to check if IRIs of individuals of a given class <$C_{context}$> correspond to a specific *IRI pattern* which must be a valid pattern argument for the SPARQL REGEX function. The IRIs of Semantic Web books, e.g., should contain the character sequence *Semantic-Web*.

Table A.37. Generic Representation of the Constraint of the Constraint Type *IRI Pattern Matching* - Applied on Subjects

context class	left p. list	right p. list	classes	c. element	c. value
Semantic-Web-Book	-	-	-	IRI pattern matching	"\bSemantic-Web\b"

Table A.38. Generic Representation of the Constraint Type *IRI Pattern Matching* - Applied on Subjects

context class	left p. list	right p. list	classes	c. element	c. value
<$C_{context}$>	-	-	-	IRI pattern matching	<IRI pattern>

Applied on predicates, constraints of this type enable to check if IRIs of object or data properties p_i $(1 \leqslant i \leqslant n)$, from which individuals of a given class <$C_{context}$> are pointing, correspond to a specific *IRI pattern* which must be a valid pattern argument for the SPARQL REGEX function.

Table A.39. Generic Representation of the Constraint Type *IRI Pattern Matching* - Applied on Predicates

context class	left p. list	right p. list	classes	c. element	c. value
<$C_{context}$>	<p_i> $(1 \leqslant i \leqslant n)$	-	-	IRI pattern matching	<IRI pattern>

Applied on objects, constraints of this type enable to check if IRIs of objects, to which individuals of a given class <$C_{context}$> are pointing via a certain object property p, correspond to a specific *IRI pattern* which must be a valid pattern argument for the SPARQL REGEX function.

Table A.40. Generic Representation of the Constraint Type *IRI Pattern Matching* - Applied on Objects

context class	left p. list	right p. list	classes	c. element	c. value
<$C_{context}$>	<p>	-	-	IRI pattern matching	<IRI pattern>

A.20 Literal Pattern Matching

Constraints of the constraint type *literal pattern matching* (*R-44*) ensure that individuals of a given class $C_{context}$ can only have relations via a specific data property p to literals of a specific datatype DT that match a certain *literal pattern*. A constraint of this type is used to validate whether all literals of a given data property p within the context of the class $C_{context}$ match a given regular expression which must be a valid pattern argument for the SPARQL REGEX function. The subsequent literal pattern matching constraint ensures

that books can only have valid *ISBN* identifiers, i.e., strings that match a given regular expression. Even though constraints of this type cannot be expressed in DL, OWL 2 can be used to formulate this constraint:

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

The first OWL 2 axiom explicitly declares *ISBN* to be a datatype. The second OWL 2 axiom defines *ISBN* as an abbreviation for a restriction on the datatype *xsd:string*. The datatype *ISBN* can be used just like any other datatype like in the universal quantification `Book` $\sqsubseteq \forall$ `isbn.ISBN`. This, all literals, to which the data property *isbn* is pointing from books, must satisfy the literal pattern matching constraint.

Table A.41. Generic Representation of the Constraint of the Constraint Type *Literal Pattern Matching*

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

Table A.42. Generic Representation of the Constraint Type *Literal Pattern Matching*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p>$	-	$<DT>$	REGEX	<literal pattern>

A.21 Negative Literal Pattern Matching

Constraints of the constraint type *negative literal pattern matching* (*R-230*) ensure that $C_{context}$ individuals cannot have relations via a given data property p to literals in the datatype DT that match a certain *literal pattern*. A constraint of this type is used to validate whether all literals of a given data property p within the context of the class $C_{context}$ do not match a given regular expression which must be a valid pattern argument for the SPARQL REGEX function. With a constraint of this type it can be ensured that an ISSN of a journal does not have be conform to a valid 10- or 13-digit ISBN, where the literal pattern for a valid ISBN is "(ISBN[-]*(1[03])*[ ]*(:)0,1)*(([0-9Xx][-]*)13—([0-9Xx][-]*)10)".

Table A.43. Generic Representation of the Constraint of the Constraint Type *Negative Literal Pattern Matching*

context class	left p. list	right p. list	classes	c. element	c. value
Journal	issn	-	xsd:string	negative literal pattern matching	<literal pattern>

Table A.44. Generic Representation of the Constraint Type *Negative Literal Pattern Matching*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	<p>	-	$<DT>$	negative literal pattern matching	<literal pattern>

A.22 Existential Quantifications

Constraints of the constraint type *existential quantifications (R-86)* (*existential restrictions* in DL) enforce that instances of given classes $C_{context}$ must have some property relation to individuals/literals of a certain class C or datatype DT via the object or data property p. An existential quantification consists of an object or data property p and a class C or a datatype DT, and defines a class $C_{context}$ that contains all those individuals that are connected by p to either (1) an individual that is an instance of C or (2) a literal that is an instance of the datatype DT. If the existential quantification is checked to ensure that each book must have at least one author and if the book *The-Hound-Of-The-Baskervilles* has no *author* relationship, then a constraint violation is raised.

$$\mathcal{K} = \{\ \texttt{Book} \sqsubseteq \exists\ \texttt{author.Person}\ \}$$

Table A.45. Generic Representation of the Constraint of the Constraint Type *Existential Quantifications*

context class	left p. list	right p. list	classes	c. element	c. value
Book	author	-	Person	existential quantification	-

Table A.46. Generic Representation of the Constraint Type *Existential Quantifications*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	<p>	-	$<C>$ | $<DT>$	existential quantification	-

A.23 Universal Quantifications

Universal quantifications (R-91) (*value restrictions* in DL) are used to enforce that all individuals of a given class $C_{context}$ are connected by particular properties p only to instances/literals of certain classes C or datatypes DT. A universal quantification consists of an object or data property p and a class C or a datatype DT, and it defines a class $C_{context}$ which contains all those individuals that are connected by p only to either (1) individuals that are instances of C or (2) literals that are instances of DT. Publications, e.g., can only have persons as authors. The triples `author(The-Lord-Of-The-Rings, Tolkien)` and `rdf:type(The-Lord-Of-The-Rings, Publication)` let a validation

engine cause a violation, in case it is not explicitly stated that *Tolkien* is a person.

$$\mathcal{K} = \{\ \texttt{Publication} \sqsubseteq \forall\ \texttt{author.Person}\ \}$$

Table A.47. Generic Representation of the Constraint of the Constraint Type *Universal Quantifications*

context class	left p. list	right p. list	classes	c. element	c. value
Publication	author	-	Person	universal quantification	-

Table A.48. Generic Representation of the Constraint Type *Universal Quantifications*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p>$	-	$<C>$ | $<DT>$	universal quantification	-

A.24 Value Restrictions

A constraint of the constraint type *value restrictions (R-88)* consists of an object or data property p and an individual i or a literal l, and it ensures that all individuals of the class $C_{context}$ are connected by p to i respectively l. Books on computer science, e.g., must have the topic *Computer-Science*.

$$\mathcal{K} = \{\ \texttt{Computer-Science-Book} \sqsubseteq \exists\ \texttt{subject.\{Computer-Science\}}\ \}$$

Table A.49. Generic Representation of the Constraint of the Constraint Type *Value Restrictions*

context class	left p. list	right p. list	classes	c. element	c. value
Computer-Science-Book	subject	-	Computer-Science	value restriction	-

Table A.50. Generic Representation of the Constraint Type *Value Restrictions*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p>$	-	$<i>$ | $<l>$	value restriction	-

A.25 Use Sub-Super Relations in Validation

Constraints of the constraint type *use sub-super relations in validation (R-224)* enforce the validation of instance data to exploit sub-class and sub-property relationships which may indicate when the data is redundant or expressed on a too general level, and therefore could be improved.

With regard to sub-super relations of properties, values of super-properties p are checked against values of each sub-property p_i $(1 \leqslant i \leqslant n)$ within the

scope of classes $C_{context}$. If a *dcterms:coverage* link is present, e.g., it could be suggested to use one of its sub-properties *dcterms:spatial* or *dcterms:temporal*. If *dcterms:date* and one of its sub-properties *dcterms:created* or *dcterms:issued* are present, e.g., it is checked if the value for *dcterms:date* is not redundant with one of the values for the sub-properties *dcterms:created* or *dcterms:issued*.

Table A.51. Generic Representation of the Constraint of the Constraint Type *Use Sub-Super Relations in Validation* - with regard to sub-super Relations of Properties

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	dcterms:date	dcterms:created, dcterms:issued	-	use sub-super relations	-

Table A.52. Generic Representation of the Constraint Type *Use Sub-Super Relations in Validation* - with regard to sub-super Relations of Properties

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p>$	$<p_i>$ $(1 \leqslant i \leqslant n)$	-	use sub-super relations	-

With regard to sub-super relations of classes, it is checked if instances of the super-class $C_{context}$ are also assigned to at least one of its sub-classes C_i $(1 \leqslant i \leqslant n)$. Publications, e.g., may be more specifically assigned to one of its sub-classes.

Table A.53. Generic Representation of the Constraint of the Constraint Type *Use Sub-Super Relations in Validation* - with regard to sub-super Relations of Classes

context class	left p. list	right p. list	classes	c. element	c. value
Publication	-	-	Book, Journal-Article, Technical-Report	use sub-super relations	-

Table A.54. Generic Representation of the Constraint Type *Use Sub-Super Relations in Validation* - with regard to sub-super Relations of Classes

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	-	-	$<C_i>$ $(1 \leqslant i \leqslant n)$	use sub-super relations	-

A.26 Negative Property Constraints

According to constraints of the constraint type *negative property constraints (R-52/53)*, instances of a specific class $C_{context}$ must not have certain object or data properties p_i $(1 \leqslant i \leqslant n)$. Books, for instance, cannot have an ISSN.

$$\mathcal{K} = \{ \ \texttt{Book} \sqsubseteq \neg(\exists \ \texttt{issn.string}) \ \}$$

Table A.55. Generic Representation of the Constraint of the Constraint Type *Negative Property Constraints*

context class	left p. list	right p. list	classes	c. element	c. value
Book	issn	-	-	negative properties	-

Table A.56. Generic Representation of the Constraint Type *Negative Property Constraints*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p_i>$ $(1 \leqslant i \leqslant n)$	-	-	negative properties	-

A.27 Language Tag Matching

For particular data properties p within the context of given classes $C_{context}$, values have to be stated for predefined languages. There must be an English variable name (*skos:notation*) for each *disco:Variable* within *disco:LogicalDataSet*s is an example of a constraint of the constraint type *language tag matching (R-47)*. Another constraint of this type restricts that there must exist a value of the data property *germanLabel* with a German language tag. The scope of the constraint includes all instances of the class *Country*. For each country, the constraint verifies that the data property, indicating its German label, points to at least one literal with a German language code.

Table A.57. Generic Representation of the Constraint of the Constraint Type *Language Tag Matching*

context class	left p. list	right p. list	classes	c. element	c. value
Country	germanLabel	-	-	language tag matching	"de"

Table A.58. Generic Representation of the Constraint Type *Language Tag Matching*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p>$	-	-	language tag matching	<language tag>

A.28 Language Tag Cardinality

For data properties, it may be desirable to restrict that values of predefined languages must be present for determined number of times. Some example constraints of the constraint type *language tag cardinality (R-48/49)* are: (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. (2) 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 [29]. (3) It should be checked if all concepts of a thesaurus have at least one common language, i.e., they have assigned at least one literal in the same language.

The constraint type *language tag cardinality* is used to restrict data properties p within the scope of classes $C_{context}$ to have a minimum, maximum,

or exact number of relationships to literals with selected language tags. The subsequent constraint of this type, e.g., enforces that there must be exactly one English title for each book.

Table A.59. Generic Representation of the Constraint of the Constraint Type *Language Tag Cardinality*

context class	left p. list	right p. list	classes	c. element	c. value
Book	title	-	-	language tag exact cardinality	"en", 1

Table A.60. Generic Representation of the Constraint Type *Language Tag Cardinality*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	<p>	-	-	language tag <minimum | maximum | exact> cardinality	<language tag>,<cardinality>

A.29 Whitespace Handling

The constraint type *whitespace handling (R-50)* can be used to avoid leading and trailing whitespaces in literals of data properties p in the context of classes $C_{context}$ by checking if literals include whitespaces and automatically delete them. This way, whitespaces of publication abstracts may automatically be deleted.

Table A.61. Generic Representation of the Constraint of the Constraint Type *Whitespace Handling*

context class	left p. list	right p. list	classes	c. element	c. value
Publication	abstract	-	-	whitespace handling	-

Table A.62. Generic Representation of the Constraint Type *Whitespace Handling*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	<p>	-	-	whitespace handling	-

A.30 HTML Handling

With constraints of the constraint type *html handling (R-51/167)*, it is possible to check (1) if there are no html tags included in literals of data properties p within the context of classes $C_{context}$ (in case the constraining value is *false*) and (2) if all HTML tags, included in literals of data properties p within the context of classes $C_{context}$, are closed properly (in case the constraining value is *true*). It is also imaginable to validate RDF in an RDF literal containing HTML with RDFa markup. As abstracts of publications, e.g., may include HTML tags, it is desirable to ensure that the HTML markup is well-formed which enables web frontends to adequately represent abstracts of publications.

Table A.63. Generic Representation of the Constraint of the Constraint Type *HTML Handling*

context class	left p. list	right p. list	classes	c. element	c. value
Publication	abstract	-	-	html handling	true

Table A.64. Generic Representation of the Constraint Type *HTML Handling*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	<p>	-	-	html handling	<boolean>

A.31 Structure

For assessing the quality of thesauri, we concentrate on the graph-based structure and apply graph- and network-analysis techniques. An example of such constraints of the constraint type *structure (R-235)* on $C_{context}$ instances 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 plain SPARQL:

```
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, ... ) ) . } }
```

Table A.65. Generic Representation of the Constraint of the Constraint Type *Structure*

context class	left p. list	right p. list	classes	c. element	c. value
skos:Concept	-	-	-	SPARQL	<SPARQL>

Table A.66. Generic Representation of the Constraint Type *Structure*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	-	-	-	SPARQL	<SPARQL>

A.32 Minimum Unqualified Cardinality Restrictions

$\geqslant np.\top$ ($\geqslant np$ in short) is a minimum unqualified cardinality restriction where $n \in \mathbb{N}$. A constraint of the constraint type *minimum unqualified cardinality restrictions (R-81)* consists of a non-negative integer n and an object or data property p, and it defines a class $C_{context}$ which contains all those individuals that are connected by p to at least n different individuals/literals. Minimum unqualified cardinality restrictions must hold for all instances of the class

$C_{context}$. The minimum unqualified cardinality restrictions constraint type can be used to ensure that books on computer science (*Computer-Science-Book*) have at least one assigned topic. In case *The-C-Programming-Language* is a computer science book without any associated subject, a constraint violation triple is generated.

$$\mathcal{K} = \{\ \texttt{Computer-Science-Book} \sqsubseteq \geqslant 1\ \texttt{subject}.\top\ \}$$

Table A.67. Generic Representation of the Constraint of the Constraint Type *Minimum Unqualified Cardinality Restrictions*

context class	left p. list	right p. list	classes	c. element	c. value
Computer-Science-Book	subject	-	$\top$	minimum cardinality	1

Table A.68. Generic Representation of the Constraint Type *Minimum Unqualified Cardinality Restrictions*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p>$	-	$\top$	minimum cardinality	$<n>$

A.33 Minimum Qualified Cardinality Restrictions

$\geqslant np.C$ is a minimum qualified cardinality restriction where $n \in \mathbb{N}$. A constraint of the constraint type *minimum qualified cardinality restrictions (R-75)* consists of a non-negative integer n, an object or data property p, and a class C or a datatype DT, and it defines a class $C_{context}$ which contains all those individuals that are connected by p to at least n different (1) individuals that are instances of C or (2) literals in DT. Minimum qualified cardinality restrictions must hold for all instances of the class $C_{context}$. The constraint type minimum qualified cardinality restrictions can be instantiated to formulate the concrete constraint that publications must have at least one author which must be a person.

$$\mathcal{K} = \{\ \texttt{Publication} \sqsubseteq \geqslant 1\ \texttt{author.Person}\ \}$$

Table A.69. Generic Representation of the Constraint of the Constraint Type *Minimum Qualified Cardinality Restrictions*

context class	left p. list	right p. list	classes	c. element	c. value
Publication	author	-	Person	minimum cardinality	1

Table A.70. Generic Representation of the Constraint Type *Minimum Qualified Cardinality Restrictions*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p>$	-	$<C>$ | $<DT>$	minimum cardinality	$<n>$

A.34 Maximum Unqualified Cardinality Restrictions

$\leqslant np.\top$ ($\leqslant np$ in short) is a maximum unqualified cardinality restriction where $n \in \mathbb{N}$. A constraint of the constraint type *maximum unqualified cardinality restrictions (R-82)* consists of a non-negative integer n and an object or data property p, and it defines a class $C_{context}$ which contains all those individuals that are connected by p to at most n different individuals/literals. Maximum unqualified cardinality restrictions must hold for all instances of the class $C_{context}$. If a person is not considered as a bestseller author, this person has sold at most 999,999 books. In other words, it should not be possible for a person that is not considered as a bestseller author having sold more than 999,999 books.

$$\mathcal{K} = \{\ \texttt{Non-Bestseller-Author} \sqsubseteq \leqslant\texttt{999,999 sellsBook}\ \}$$

Table A.71. Generic Representation of the Constraint of the Constraint Type *Maximum Unqualified Cardinality Restrictions*

context class	left p. list	right p. list	classes	c. element	c. value
Non-Bestseller-Author	sellsBook	-	$\top$	maximum cardinality	999,999

Table A.72. Generic Representation of the Constraint Type *Maximum Unqualified Cardinality Restrictions*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p>$	-	$\top$	maximum cardinality	$<n>$

A.35 Maximum Qualified Cardinality Restrictions

$\leqslant np.C$ is a maximum qualified cardinality restriction where $n \in \mathbb{N}$. A constraint of the constraint type *maximum qualified cardinality restrictions (R-76)* consists of a non-negative integer n, an object or data property p, and a class C or a datatype DT, and it defines a class $C_{context}$ which contains all those individuals that are connected by p to at most n different (1) individuals that are instances of C or (2) literals in DT. Maximum qualified cardinality restrictions must hold for all instances of the class $C_{context}$. Children can have at most 2 parents is an example constraint of the type maximum qualified cardinality restrictions.

$$\mathcal{K} = \{\ \texttt{Child} \sqsubseteq \leqslant\texttt{2 childOf.Parent}\ \}$$

Table A.73. Generic Representation of the Constraint of the Constraint Type *Maximum Qualified Cardinality Restrictions*

context class	left p. list	right p. list	classes	c. element	c. value
Child	childOf	-	Parent	maximum cardinality	2

Table A.74. Generic Representation of the Constraint Type *Maximum Qualified Cardinality Restrictions*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p>$	-	$<C>$ | $<DT>$	maximum cardinality	$<n>$

A.36 Exact Unqualified Cardinality Restrictions

$\geqslant np.\top \sqcap \leqslant np.\top$ is an exact unqualified cardinality restriction where $n \in \mathbb{N}$. A constraint of the constraint type *exact unqualified cardinality restrictions (R-80)* consists of a non-negative integer n and an object or data property p, and it defines a class $C_{context}$ which contains all those individuals that are connected by p to exactly n different individuals/literals. Exact unqualified cardinality restrictions must hold for all instances of the class $C_{context}$. An author of a one-hit wonder book can only be an author having written exactly one successful book is an example constraint of the constraint type exact unqualified cardinality restrictions.

$\mathcal{K}$ = {
One-Hit-Wonder-Book-Author $\sqsubseteq$
$\geqslant$1 author-Of-Successful-Book $\sqcap$ $\leqslant$1 author-Of-Successful-Book }

Table A.75. Generic Representation of the Constraint of the Constraint Type *Exact Unqualified Cardinality Restrictions*

context class	left p. list	right p. list	classes	c. element	c. value
One-Hit-Wonder-Book-Author	author-Of-Successful-Book	-	$\top$	exact cardinality	1

Table A.76. Generic Representation of the Constraint Type *Exact Unqualified Cardinality Restrictions*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p>$	-	$\top$	exact cardinality	$<n>$

A.37 Exact Qualified Cardinality Restrictions

$\geqslant np.C \sqcap \leqslant np.C$ is an exact qualified cardinality restriction where $n \in \mathbb{N}$. A constraint of the constraint type *exact qualified cardinality restrictions (R-74)* consists of a non-negative integer n, an object or data property p, and a class C or a datatype DT, and it defines a class $C_{context}$ which contains all those individuals that are connected by p to exactly n different (1) individuals that are instances of C or (2) literals in DT. Exact qualified cardinality restrictions must hold for all instances of the class $C_{context}$. Every person, e.g., is a child of exactly two parents.

$\mathcal{K}$ = { Person $\sqsubseteq$ $\geqslant$2 childOf.Parent $\sqcap$ $\leqslant$2 childOf.Parent }

Table A.77. Generic Representation of the Constraint of the Constraint Type *Exact Qualified Cardinality Restrictions*

context class	left p. list	right p. list	classes	c. element	c. value
Person	childOf	-	Parent	exact cardinality	2

Table A.78. Generic Representation of the Constraint Type *Exact Qualified Cardinality Restrictions*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p>$	-	$<C>$ | $<DT>$	exact cardinality	$<n>$

A.38 Cardinality Shortcuts

In most library applications, *cardinality shortcuts (R-228)* tend to appear in pairs, with *optional / mandatory* establishing minimum cardinality restrictions on properties p_i $(1 \leqslant i \leqslant n)$ within the context of classes $C_{context}$ and *repeatable / non-repeatable* for maximum cardinality restrictions (see Table A.79 for the possible pairs of cardinality shortcuts). For publications, e.g., information about their authors should be mandatory and repeatable.

Table A.79. Pairs of Cardinality Shortcuts

cardinality shortcuts	[min,max]
optional & non-repeatable	[0,1]
optional & repeatable	[0,*]
mandatory & non-repeatable	[1,1]
mandatory & repeatable	[1,*]

Table A.80. Generic Representation of the Constraint of the Constraint Type *Cardinality Shortcuts*

context class	left p. list	right p. list	classes	c. element	c. value
Publication	author	-	-	mandatory and repeatable properties	-

Table A.81. Generic Representation of the Constraint Type *Cardinality Shortcuts*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p_i>$ $(1 \leqslant i \leqslant n)$	-	-	<cardinality shortcut pair>	-

A.39 Vocabulary

The constraint type *vocabulary (R-220)* serves to ensure that for $C_{context}$ individuals neither deprecated vocabulary terms are used nor elements which are not specified in listed namespaces. Constraints of this type are easier to check in case IRIs of used vocabulary terms are dereferenceable. The next concrete constraint of this type enables to check if each skos:Concept of the

input graph is connected to resources/literals via properties which are actually defined within current versions of used vocabulary namespaces.

Table A.82. Generic Representation of the Constraint of the Constraint Type *Vocabulary*

context class	left p. list	right p. list	classes	c. element	c. value
skos:Concept	-	-	-	vocabulary	-

Table A.83. Generic Representation of the Constraint Type *Vocabulary*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	-	-	-	vocabulary	-

A.40 Provenance

According to constraints of the constraint type *provenance (R-24)*, some provenance information like *dcterms:provenance* must be available for instances of a given class $C_{context}$. Series, studies, data sets, data files, or publications, for instance, should have provenance information associated with them.

Table A.84. Generic Representation of the Constraint of the Constraint Type *Provenance*

context class	left p. list	right p. list	classes	c. element	c. value
Publication	-	-	-	provenance	-

Table A.85. Generic Representation of the Constraint Type *Provenance*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	-	-	-	provenance	-

A.41 Required Properties

Resources of a given type $C_{context}$ sometimes must be accompanied by specified mandatory object or data properties p_i $(1 \leqslant i \leqslant n)$. Places must have the properties latitude and longitude, persons must have a birth date, organizations must have a name, and publications must have a title are example constraints of the constraint type *required properties (R-68)*.

$$\mathcal{K} = \{ \texttt{ Publication} \sqsubseteq \exists \texttt{ title.string} \}$$

Table A.86. Generic Representation of the Constraint of the Constraint Type *Required Properties*

context class	left p. list	right p. list	classes	c. element	c. value
Publication	title	-	-	required properties	-

Table A.87. Generic Representation of the Constraint Type *Required Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p_i>$ $(1 \leqslant i \leqslant n)$	-	-	required properties	-

A.42 Optional Properties

Constraints of the constraint type *optional properties (R-69)* serve to define that it is optional to link instances of the class $C_{context}$ via object or data properties p_i $(1 \leqslant i \leqslant n)$ to any other resources or literals. For instance, it should be optional to state a DOI for books.

$$\mathcal{K} = \{ \quad \exists \ \texttt{doi.string} \sqsubseteq \texttt{Book} \ \}$$

Table A.88. Generic Representation of the Constraint of the Constraint Type *Optional Properties*

context class	left p. list	right p. list	classes	c. element	c. value
Book	doi	-	-	optional properties	-

Table A.89. Generic Representation of the Constraint Type *Optional Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p_i>$ $(1 \leqslant i \leqslant n)$	-	-	optional properties	-

A.43 Repeatable Properties

With constraints of the constraint type *repeatable properties (R-70)*, one can formulate that for individuals of classes $C_{context}$ object or data properties p_i $(1 \leqslant i \leqslant n)$ are repeatable, i.e., their cardinality is at least one. The object property *author*, e.g., is repeatable for individuals of the class *Publication*.

$$\mathcal{K} = \{ \quad \texttt{Publication} \sqsubseteq \geqslant 1 \ \texttt{author}.\top \ \}$$

Table A.90. Generic Representation of the Constraint of the Constraint Type *Repeatable Properties*

context class	left p. list	right p. list	classes	c. element	c. value
Publication	author	-	-	repeatable properties	-

Table A.91. Generic Representation of the Constraint Type *Repeatable Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p_i>$ $(1 \leqslant i \leqslant n)$	-	-	repeatable properties	-

A.44 Conditional Properties

The constraint type *conditional properties (R-71)* is used to specify for $C_{context}$ instances that if the properties p_i $(1 \leqslant i \leqslant n)$ are present, then the properties p_j $(1 \leqslant j \leqslant n)$ also have to be present. If an individual has a *parentOf* relationship, then this individual must also have an *ancestorOf* relation is an example of a constraint of this type.

$$\mathcal{K} = \{ \ \texttt{parentOf} \sqsubseteq \texttt{ancestorOf} \ \}$$

Table A.92. Generic Representation of the Constraint of the Constraint Type *Conditional Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	parentOf	ancestorOf	-	conditional properties	-

Table A.93. Generic Representation of the Constraint Type *Conditional Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p_i>$ $(1 \leqslant i \leqslant n)$	$<p_j>$ $(1 \leqslant j \leqslant n)$	-	conditional properties	-

A.45 Recommended Properties

Optional properties p_i $(1 \leqslant i \leqslant n)$ can be marked as recommended within a particular context $C_{context}$ so that a report of missing recommended properties may be generated. An example of a constraint of the constraint type *recommended properties (R-72)* is to recommend to specify the number of pages for publications.

Table A.94. Generic Representation of the Constraint of the Constraint Type *Recommended Properties*

context class	left p. list	right p. list	classes	c. element	c. value
Publication	numberOfPages	-	-	recommended properties	-

Table A.95. Generic Representation of the Constraint Type *Recommended Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p_i>$ $(1 \leqslant i \leqslant n)$	-	-	recommended properties	-

A.46 Severity Levels

A concrete constraint is instantiated from one of the 81 constraint types and is 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. We use the classification system of log messages in software development like *Apache Log4j 2* [3], the *Java Logging API*,[5] and the *Apache Commons Logging API*[6] 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* are 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. Although default severity levels should be provided for each constraint to indicate how serious the violation of the constraint is within the context of a specific vocabulary, validation environments should enable users to adapt the severity levels of constraints according to their individual needs.

With constraints of the constraint type *severity levels (R-158)*, one can associate certain severity levels with particular classes $C_{context}$ to indicate that individuals of these classes always cause constraint violations of a certain kind of severity. For instances of the class *High-Severity*, e.g., the severity of constraint violations should always be high independently of the severity of violated constraints. Such a constraint makes sense for all those instances which are considered to be essential within the data to be validated.

Table A.96. Generic Representation of the Constraint of the Constraint Type *Severity Levels*

context class	left p. list	right p. list	classes	c. element	c. value
High-Severity	-	-	-	severity level	"error"

Table A.97. Generic Representation of the Constraint Type *Severity Levels*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	-	-	-	severity level	<severity level>

A.47 Labeling and Documentation

It should be possible to provide human-readable labels (intended for human consumption such as documentation or user interfaces) for (1) a property not just globally but also within the context of a class $C_{context}$ or (2) a class $C_{context}$ under certain conditions expressible in SPARQL. This way, human-readable documentation can be offered for first order elements of an application profile. The goal is to provide a documentation or comments area that is intended for human readers where a description of a property or a class

[5] http://docs.oracle.com/javase/7/docs/api/java/util/logging/Level.html
[6] http://commons.apache.org/proper/commons-logging/

can be made. This could both define a property or a class as well as include other information such as the date of its publication.

Example constraints of the constraint type *labeling and documentation (R-208)*, against which $C_{context}$ individuals are validated, are: (1) Undocumented concepts: The SKOS standard defines a number of properties which are useful for documenting the meaning of concepts in a thesaurus in a human-readable form. An intense use of these properties leads to a well-documented thesaurus which should also improve its quality. (2) Overlapping labels: Two concepts having the same preferred lexical label in a given language when they belong to the same concept scheme, could indicate missing disambiguation information and thus lead to problems in auto-completion applications. (3) Missing labels: To make a vocabulary more convenient for humans to use, instances of SKOS classes (skos:Concept, skos:ConceptScheme, skos:Collection) should be labeled using, e.g., skos:prefLabel, skos:altLabel, rdfs:label, and dc:title. (4) Concepts within the same concept scheme should not have identical skos:notation literals:

```
SPARQL:
SELECT ?subject WHERE {
    ?subject
        a skos:Concept ;
        skos:inScheme ?conceptScheme ;
        skos:notation ?l .
    ?concept
        a skos:Concept ;
        skos:inScheme ?conceptScheme ;
        skos:notation ?l .
    FILTER ( ?subject != ?concept ) . } }
```

As the complexity of constraints of this type is high in general, they have to be expressed using plain SPARQL.

Table A.98. Generic Representation of the Constraint of the Constraint Type *Labeling and Documentation*

context class	left p. list	right p. list	classes	c. element	c. value
skos:Concept	-	-	-	SPARQL	<SPARQL>

Table A.99. Generic Representation of the Constraint Type *Labeling and Documentation*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	-	-	-	SPARQL	<SPARQL>

A.48 Context-Specific Property Groups

Constraints of the constraint type *context-specific property groups (R-66)* enforce that instances within a given context must have relations of all properties p_i ($1 \leqslant i \leqslant n$) defined within a group of properties. The context may be

an application profile, a shape, or in most cases a class $C_{context}$. A constraint of this type consists of a list of object or data properties p_i $(1 \leqslant i \leqslant n)$ and defines a class $C_{context}$ that contains all those individuals that are connected by each of these properties p_i to resources/literals. A book, e.g., must have an *isbn* and a *title* link.

$$\mathcal{K} = \{\ \texttt{Book} \equiv \exists\ \texttt{isbn.string} \sqcap \exists\ \text{title.string}\ \}$$

Table A.100. Generic Representation of the Constraint of the Constraint Type *Context-Specific Property Groups*

context class	left p. list	right p. list	classes	c. element	c. value
Book	isbn, title	-	-	property group	-

Table A.101. Generic Representation of the Constraint Type *Context-Specific Property Groups*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p_i>$ $(1 \leqslant i \leqslant n)$	-	-	property group	-

A.49 Context-Specific Exclusive OR of Properties

Exclusive or is a logical operation that outputs true whenever both inputs differ (one is true, the other is false). Constraints of the constraint type *context-specific exclusive OR of properties (R-11)* restrict individuals within a specific context to have exactly one property relationship of multiple mutually exclusive properties p_i $(1 \leqslant i \leqslant n)$. The context may be an application profile, a shape, or in most cases a class $C_{context}$. Publications, e.g., are either identified by an ISBN (for books) *or* by an ISSN (for periodical publications), but it should not be possible to assign both identifiers to a given publication. In case *Moby-Dick* is a publication with an ISBN without an ISSN, *Moby-Dick* is considered as a valid publication.

$$\begin{aligned}&\mathcal{K} = \{\\ &\texttt{Publication} \sqsubseteq (\neg\texttt{A} \sqcap \texttt{B}) \sqcup (\texttt{A} \sqcap \neg\texttt{B}),\\ &\texttt{A} \sqsubseteq \geqslant 1\ \texttt{isbn.string} \sqcap \leqslant 1\ \texttt{isbn.string},\\ &\texttt{B} \sqsubseteq \geqslant 1\ \texttt{issn.string} \sqcap \leqslant 1\ \texttt{issn.string}\ \}\end{aligned}$$

Table A.102. Generic Representation of the Constraint of the Constraint Type *Context-Specific Exclusive OR of Properties*

context class	left p. list	right p. list	classes	c. element	c. value
Publication	isbn, issn	-	-	exclusive or	-

Table A.103. Generic Representation of the Constraint Type *Context-Specific Exclusive OR of Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p_i>$ $(1 \leqslant i \leqslant n)$	-	-	exclusive or	-

A.50 Context-Specific Exclusive OR of Property Groups

Exclusive or is a logical operation that outputs true whenever both inputs differ (one is true, the other is false). Constraints of the constraint type *context-specific exclusive OR of property groups (R-13)* restrict individuals within a specific context to have property relationships of all properties p_i $(1 \leqslant i \leqslant n)$ defined within exactly one of multiple mutually exclusive property groups C_i $(1 \leqslant i \leqslant n)$. The context may be an application profile, a shape, or in most cases a class $C_{context}$. Publications, e.g., are either identified by an ISBN and a title (for books) *or* by an ISSN and a title (for periodical publications), but it should not be possible to assign both identifiers to a given publication. In case *The-Great-Gatsby* is a publication with an ISBN and a title without an ISSN, *The-Great-Gatsby* is considered as a valid publication. The subsequent DL statements demonstrate that the constraint is complex as it is composed of many other simple constraints (*minimum (R-75) and maximum qualified cardinality restrictions (R-76)*, *intersection (R-15/16)*, *disjunction (R-17/18)*, and *negation (R-19/20)*).

$$
\begin{aligned}
&\mathcal{K} = \{\\
&\texttt{Publication} \sqsubseteq (\neg \texttt{E} \sqcap \texttt{F}) \sqcup (\texttt{E} \sqcap \neg \texttt{F}),\\
&\texttt{E} \equiv \texttt{A} \sqcap \texttt{B} \ , \ \texttt{F} \equiv \texttt{C} \sqcap \texttt{D},\\
&\texttt{A} \sqsubseteq \geqslant 1\ \texttt{isbn.string} \sqcap \leqslant 1\ \texttt{isbn.string},\\
&\texttt{B} \sqsubseteq \geqslant 1\ \texttt{title.string} \sqcap \leqslant 1\ \texttt{title.string},\\
&\texttt{C} \sqsubseteq \geqslant 1\ \texttt{issn.string} \sqcap \leqslant 1\ \texttt{issn.string},\\
&\texttt{D} \sqsubseteq \geqslant 1\ \texttt{title.string} \sqcap \leqslant 1\ \texttt{title.string}\ \}
\end{aligned}
$$

Table A.104. Generic Representation of the Constraint of the Constraint Type *Context-Specific Exclusive OR of Property Groups*

context class	left p. list	right p. list	classes	c. element	c. value
Publication	-	-	E, F	exclusive or	-
E	-	-	A, B	intersection	-
F	-	-	C, D	intersection	-
A	isbn	-	string	exact cardinality	1
B	title	-	string	exact cardinality	1
C	issn	-	string	exact cardinality	1
D	title	-	string	exact cardinality	1

Table A.105. Generic Representation of the Constraint Type *Context-Specific Exclusive OR of Property Groups*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	-	-	$<C_i>$ $(1 \leqslant i \leqslant n)$	exclusive or	-

A.51 Context-Specific Inclusive OR of Properties

Inclusive or is a logical connective joining two or more predicates that yields the logical value *true* when at least one of the predicates is true. Constraints of the constraint type *context-specific inclusive OR of properties (R-202)* are used to check if individuals within a specific context match at least one of multiple inclusive properties, i.e., to ensure that individuals within that context must have at least one property relationship of listed properties p_i ($1 \leqslant i \leqslant n$). The context may be an application profile, a shape, or in most cases a class $C_{context}$. A person, e.g., must have either exactly one name *or* exactly one given name. In contrast to an exclusive or of properties, a person should also be considered to be valid if both a name and a given name are stated.

$$
\begin{aligned}
&\mathcal{K} = \{ \\
&\texttt{Person} \sqsubseteq \texttt{A} \sqcup \texttt{B}, \\
&\texttt{A} \sqsubseteq \geqslant 1\ \texttt{name.string} \sqcap \leqslant 1\ \texttt{name.string}, \\
&\texttt{B} \sqsubseteq \geqslant 1\ \texttt{givenName.string} \sqcap \leqslant 1\ \texttt{givenName.string}\ \}
\end{aligned}
$$

Table A.106. Generic Representation of the Constraint of the Constraint Type *Context-Specific Inclusive OR of Properties*

context class	left p. list	right p. list	classes	c. element	c. value
Person	name, givenName	-	-	inclusive or	-

Table A.107. Generic Representation of the Constraint Type *Context-Specific Inclusive OR of Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p_i>$ $(1 \leqslant i \leqslant n)$	-	-	inclusive or	-

A.52 Context-Specific Inclusive OR of Property Groups

Inclusive or is a logical connective joining two or more predicates that yields the logical value *true* when at least one of the predicates is true. Constraints of the constraint type *context-specific inclusive OR of property groups (R-227)* are used to check if individuals within a specific context match at least one of a set of inclusive property groups, i.e., to ensure that individuals within that context must have all property links p_i ($1 \leqslant i \leqslant n$) of at least one of the inclusive property groups C_i ($1 \leqslant i \leqslant n$). The context may be an application profile, a shape, or in most cases a class $C_{context}$. A person, e.g., must have either exactly one name *or* at least one given name and exactly one family name. In contrast to an exclusive or of property groups, a person should also be considered to be valid if, e.g., a name, a given name, and a family name are stated.

$$\mathcal{K} = \{$$
$$\texttt{Person} \sqsubseteq \texttt{A} \sqcup \texttt{B},$$
$$\texttt{A} \sqsubseteq \geqslant 1\ \texttt{name.string} \sqcap \leqslant 1\ \texttt{name.string},$$
$$\texttt{B} \equiv \texttt{C} \sqcap \texttt{D},$$
$$\texttt{C} \sqsubseteq \geqslant 1\ \texttt{givenName.string},$$
$$\texttt{D} \sqsubseteq \geqslant 1\ \texttt{familyName.string} \sqcap \leqslant 1\ \texttt{familyName.string}\ \}$$

Table A.108. Generic Representation of the Constraint of the Constraint Type *Context-Specific Inclusive OR of Property Groups*

context class	left p. list	right p. list	classes	c. element	c. value
Person	-	-	A, B	inclusive or	-
A	name	-	string	exact cardinality	1
B	-	-	C, D	intersection	-
C	givenName	-	string	minimum cardinality	1
D	familyName	-	string	exact cardinality	1

Table A.109. Generic Representation of the Constraint Type *Context-Specific Inclusive OR of Property Groups*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	-	-	$<C_i>$ $(1 \leqslant i \leqslant n)$	inclusive or	-

A.53 Mathematical Operations

Constraints of the constraint type *mathematical operations (R-41/42)* enable to check for individuals of a particular class $C_{context}$ if the outcomes of mathematical operations are correct. Mathematical operations are executed on literals of a certain datatype DT, the output argument of mathematical operations is the literal of the data property p, and the input arguments are literals of the data properties p_i $(1 \leqslant i \leqslant n)$.

Example constraints of this constraint type are to check the addition of date literals, the addition of days to a start date, or statistical computations like average, mean, and sum. For a given variable, e.g., the sum of *valid cases* and *invalid cases* has to be equal to the *total number of cases*. For rectangles, the literal of the data property *area* must exactly be the outcome of the multiplication of literals of the data properties *width* and *height*, whereby the multiplication is performed on literals of the datatype *xsd:nonNegativeInteger*.

Table A.110. Generic Representation of the Constraint of the Constraint Type *Mathematical Operations*

context class	left p. list	right p. list	classes	c. element	c. value
Rectangle	area	width, height	xsd:nonNegativeInteger	multiplication	-

Table A.111. Generic Representation of the Constraint Type *Mathematical Operations*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p>$	$<p_i>$ $(1 \leqslant i \leqslant n)$	$<DT>$	<mathematical operation>	-

A.54 Ordering

With constraints of the constraint type *ordering (R-121/217)*, objects of the class C or literals of the datatype DT can be declared to be ordered for given properties p within the context of given classes $C_{context}$. If objects should be ordered, they must be represented as *skos:Concept*s and be members (*skos:memberList*) in an ordered collection (*skos:OrderedCollection*).

Variables, questions, codes, and categories, e.g., are typically organized in a particular order. If disco:Questions of a given disco:Questionnaire should be ordered, a collection of questions, the questionnaire, must be of the type skos:OrderedCollection which may contain multiple questions - each represented as skos:Concept in a skos:memberList. If codes/categories of a given disco:Representation of a given disco:Variable should be ordered, the variable representation must be of the type skos:OrderedCollection which may contain multiple codes/categories - each represented as skos:Concept in a skos:memberList. If disco:Variables of a given disco:LogicalDataSet should be ordered, a collection of variables must be of the type skos:OrderedCollection which may contain multiple variables - each represented as skos:Concept in a skos:memberList.

Table A.112. Generic Representation of the Constraint of the Constraint Type *Ordering*

context class	left p. list	right p. list	classes	c. element	c. value
LogicalDataSet	variable	-	Variable	ordered values	-

Table A.113. Generic Representation of the Constraint Type *Ordering*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p>$	-	$<C>$ | $<DT>$	ordered values	-

A.55 Inverse Object Properties

In many cases, properties are used bi-directionally and then accessed in the inverse direction, e.g., parent $\equiv$ child$^-$. There should be a way to declare the value type and the cardinality of those inverse relations without having to declare a new property IRI. A constraint of the constraint type *inverse object properties (R-56)* states that the object property p_1 is an inverse of the object property p_2. Thus, if an individual i_1 is connected by p_1 to an individual i_2, then i_2 is also connected by p_2 to i_1, and vice versa. The object property *authorOf*, e.g., is an inverse of the object property *author*.

$$\mathcal{K} = \{\ author \equiv authorOf^-\ \}$$

Table A.114. Generic Representation of the Constraint of the Constraint Type *Inverse Object Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	author	authorOf	-	inverse property	-

Table A.115. Generic Representation of the Constraint Type *Inverse Object Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	$<p_1>$	$<p_2>$	-	inverse property	-

A.56 Symmetric Object Properties

A property is symmetric if it is equivalent to its own inverse [27]. A constraint of the constraint type *symmetric object properties (R-61)* states that the object property p is symmetric - that is, if an individual i_1 is connected by p to an individual i_2, then i_2 is also connected by p to i_1. The object property *coAuthorOf* is symmetric.

$$\mathcal{K} = \{\ coAuthorOf \equiv coAuthorOf^{-}\ \}$$

Table A.116. Generic Representation of the Constraint of the Constraint Type *Symmetric Object Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	coAuthorOf	-	-	symmetric property	-

Table A.117. Generic Representation of the Constraint Type *Symmetric Object Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	$<p>$	-	-	symmetric property	-

A.57 Asymmetric Object Properties

A property is asymmetric if it is disjoint from its own inverse [27]. A constraint of the constraint type *asymmetric object properties (R-62)* states that the object property p is asymmetric - that is, if an individual i_1 is connected by p to an individual i_2, then i_2 cannot be connected by p to i_1. The object property *author* is asymmetric.

$$\mathcal{K} = \{\ author \sqcap author^{-} \sqsubseteq \bot\ \}$$

Table A.118. Generic Representation of the Constraint of the Constraint Type *Asymmetric Object Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	author	-	-	asymmetric property	-

Table A.119. Generic Representation of the Constraint Type *Asymmetric Object Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	$<p>$	-	-	asymmetric property	-

A.58 Transitive Object Properties

Transitivity in DL is a special form of complex role inclusion. A constraint of the constraint type *transitive object properties (R-63)* states that the object property p is transitive - that is, if an individual i_1 is connected by p to an individual i_2 that is connected by p to an individual i_3, then i_1 is also connected by p to i_3. The object property *ancestorOf* is an example of a transitive object property.

$$\mathcal{K} = \{\ \texttt{ancestorOf} \circ \texttt{ancestorOf} \sqsubseteq \texttt{ancestorOf}\ \}$$

Table A.120. Generic Representation of the Constraint of the Constraint Type *Transitive Object Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	ancestorOf	-	-	transitive property	-

Table A.121. Generic Representation of the Constraint Type *Transitive Object Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	$<p>$	-	-	transitive property	-

A.59 Self Restrictions

A constraint of the constraint type *self restrictions (R-89)* consists of an object property p, and it makes sure that all individuals of the class $C_{context}$ are connected by p to themselves. If authors are known to always cite themselves, i.e., they are assigned to the class *Citing-Themselves*, they must be linked to themselves via the object property *cite*.

$$\mathcal{K} = \{\ \texttt{Citing-Themselves} \sqsubseteq \exists\ \texttt{cite.Self}\ \}$$

Table A.122. Generic Representation of the Constraint of the Constraint Type *Self Restrictions*

context class	left p. list	right p. list	classes	c. element	c. value
Citing-Themselves	cite	-	-	self restriction	-

Table A.123. Generic Representation of the Constraint Type *Self Restrictions*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p>$	-	-	self restriction	-

A.60 Valid Identifiers

With constraints of the constraint type *valid identifiers (R-98/171)* the validity of URIs/IRIs of $C_{context}$ resources is checked by dereferencing which could be done from a syntactical, semantic, or functional point of view. We keep the functional aspect, which means to check whether HTTP URIs are dereferenceable (without HTTP 404 errors) or not. A reference can be (1) a real HTTP URI redirecting to a LOD resource, (2) an HTTP URI that is not dereferenceable, (3) a local URI that is not dereferenceable, or (4) an identifier that is not dereferenceable. Dereferencing URIs returns either a resource or an error in case an HTTP error code occurs. In the context of Linked Data, we restrict ourselves to using HTTP URIs/IRIs only and avoid other URI schemes such as URNs and DOIs.

Table A.124. Generic Representation of the Constraint of the Constraint Type *Valid Identifiers*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	-	-	-	valid identifiers	-

Table A.125. Generic Representation of the Constraint Type *Valid Identifiers*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	-	-	-	valid identifiers	-

A.61 Recursive Queries

ReSh and ShEx are recursive constraint languages, i.e., the value shape of a resource shape is in turn another resource shape. There is no way to express that in SPARQL as SPARQL can't express recursive queries, e.g., to test if a publication and all referenced publications are valid. Nevertheless, most SPARQL engines already have functions that go beyond the official SPARQL 1.1 specification. In addition, [30] recently proposed an extension operator to SPARQL to include recursion. A publication ($C_{context}$), e.g., may reference (p) another publication which may also reference another publication and so

forth is an example of a constraint of the constraint type *recursive queries (R-222)*, since the referenced publications are recursively validated as well.

```
1 ShEx:
2 Publication {
3     references @Publication* }
```

Table A.126. Generic Representation of the Constraint of the Constraint Type *Recursive Queries*

context class	left p. list	right p. list	classes	c. element	c. value
Publication	references	-	-	recursive	-

Table A.127. Generic Representation of the Constraint Type *Recursive Queries*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p>$	-	-	recursive	-

A.62 Reflexive Object Properties

The constraint type *reflexive object properties (R-59)* corresponds to *reflexive roles* or *global reflexivity* in DL. Global reflexivity can be expressed by imposing local reflexivity on the top concept [27]. A constraint of the constraint type reflexive object properties states that the object property p is reflexive - that is, each individual is connected by p to itself. Each individual knows itself is a reflexive object properties constraint.

$$\mathcal{K} = \{\ \top \sqsubseteq \exists\ \texttt{knows.Self}\ \}$$

Table A.128. Generic Representation of the Constraint of the Constraint Type *Reflexive Object Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	knows	-	-	reflexive property	-

Table A.129. Generic Representation of the Constraint Type *Reflexive Object Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	$<p>$	-	-	reflexive property	-

A.63 Class-Specific Reflexive Object Properties

In DL terminology, the constraint type *class-specific reflexive object properties (R-231)* is called *local reflexivity* - a set of instances of a specific concept

that are related to themselves via a given role [27]. A constraint of this constraint type states that the object property p is reflexive within the context of a particular class $C_{context}$ - that is, each individual of the class $C_{context}$ is connected by p to itself. The class *Talking-To-Themselves*, e.g., denotes the set of individuals that are talking to themselves.

$$\mathcal{K} = \{ \texttt{Talking-To-Themselves} \sqsubseteq \exists \texttt{ talksTo.Self} \}$$

Table A.130. Generic Representation of the Constraint of the Constraint Type *Class-Specific Reflexive Object Properties*

context class	left p. list	right p. list	classes	c. element	c. value
Talking-To-Themselves	talksTo	-	-	reflexive property	-

Table A.131. Generic Representation of the Constraint Type *Class-Specific Reflexive Object Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p>$	-	-	reflexive property	-

A.64 Irreflexive Object Properties

A property is irreflexive if it is never locally reflexive [27]. Constraints of the *irreflexive object properties (R-60)* constraint type (*irreflexive roles* in DL) state that the object property p is irreflexive - that is, no individual is connected by p to itself. With the irreflexive object property constraint $\top \sqsubseteq \neg\exists authorOf.Self$, e.g., one can state that individuals cannot be authors of themselves.

$$\mathcal{K} = \{ \top \sqsubseteq \neg\exists authorOf.Self \}$$

Table A.132. Generic Representation of the Constraint of the Constraint Type *Irreflexive Object Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	authorOf	-	-	irreflexive property	-

Table A.133. Generic Representation of the Constraint Type *Irreflexive Object Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	$<p>$	-	-	irreflexive property	-

A.65 Class-Specific Irreflexive Object Properties

A property is irreflexive if it is never locally reflexive [27]. A constraint of the constraint type *class-specific irreflexive object properties (R-232)* states

that the object property p is irreflexive within the context of a given class $C_{context}$ - that is, no individual of the class $C_{context}$ is connected by p to itself. Persons, e.g., cannot be married to themselves.

$$\mathcal{K} = \{\ \texttt{Person} \sqsubseteq \neg\ \exists marriedTo.Self\ \ \}$$

Table A.134. Generic Representation of the Constraint of the Constraint Type *Class-Specific Irreflexive Object Properties*

context class	left p. list	right p. list	classes	c. element	c. value
Person	marriedTo	-	-	irreflexive property	-

Table A.135. Generic Representation of the Constraint Type *Class-Specific Irreflexive Object Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p>$	-	-	irreflexive property	-

A.66 Data Model Consistency

The purpose of *data model consistency (R-234)* constraints is to ensure the integrity of data according to the intended semantics of vocabularies' data models. Constraints of this type are evaluated for each individual of the class $C_{context}$. Every *qb:Observation*, e.g., must have a value for each dimension declared in its *qb:DataStructureDefinition* and no two *qb:Observations* in the same *qb:DataSet* can have the same value for all dimensions. 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. As the complexity of constraints of this type is high in general, they have to be expressed using plain SPARQL.

Table A.136. Generic Representation of the Constraint of the Constraint Type *Data Model Consistency*

context class	left p. list	right p. list	classes	c. element	c. value
qb:DataSet	-	-	-	SPARQL	<SPARQL>

Table A.137. Generic Representation of the Constraint Type *Data Model Consistency*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	-	-	-	SPARQL	<SPARQL>

A.67 Handle RDF Collections

RDF collections can be used in data as well as in constraint expressions. Example constraints of the *handle rdf collections (R-120)* constraint type are:

(1) all elements of a collection must be of a specific class or datatype, (2) the first/last element of a given list must be a specific literal or object, (3) elements of collections are compared in a certain way, (4) collections must be identical, (5) actions on lists like retrieving the nth member or all members are automatically executed, (6) the i. list element must be equal to a particular value, (7) a given collection must have more than, less than, or exactly a determined number of elements. With the subsequent constraint of this type, the element "Computer Science" of the datatype *xsd:string* (C | DT) is automatically added to the list the property *topics* (p) is pointing to from instances of the class *Computer-Science-Publication* ($C_{context}$).

Table A.138. Generic Representation of the Constraint of the Constraint Type *Handle RDF Collections*

context class	left p. list	right p. list	classes	c. element	c. value
Computer-Science-Publication	topics	-	xsd:string	add element to list	"Computer Science"

Table A.139. Generic Representation of the Constraint Type *Handle RDF Collections*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p>$	-	$<C>$ | $<DT>$	<action on RDF collection>	<argument>

A.68 Membership in Controlled Vocabularies

In some cases, resources must be members of listed controlled vocabularies. Constraints of the constraint type *membership in controlled vocabularies* (*R-32*) guarantee that individuals of a given class $C_{context}$ are assigned to the class *skos:Concept* and are included in at least one of possibly multiple controlled vocabularies. In other words, these skos:Concepts can only be related to controlled vocabularies contained in a list of allowed controlled vocabularies i_i ($1 \leqslant i \leqslant n$) via the object properties *skos:inScheme* and/or *skos:hasTopConcept*. Furthermore, listed controlled vocabularies must be assigned to the class *skos:ConceptScheme*.

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. Resources of the type disco:SummaryStatistics, e.g., can only have disco:summaryStatisticType relationships to skos:Concepts which must be members of the controlled vocabulary ddicv:SummaryStatisticType. Objects like *Computer-Science*, *Informatics*, and *Information-Technology*, to which the object property *subject* is pointing, e.g., have to be of the class skos:Concept and must be contained in at least one of the allowed controlled vocabularies *Computer-Science-Book-Subjects*, *Computer-Science-Book-Topics*, or *Computer-Science-Book-Categories*.

$\mathcal{K}$ = {
Computer-Science-Book-Subject ⊑ Concept ⊓ ∀ inScheme.Controlled-Vocabulary,
Controlled-Vocabulary ⊑ ConceptScheme ⊓ ({Computer-Science-Book-Subjects} ⊔
{Computer-Science-Book-Topics} ⊔ {Computer-Science-Book-Categories}) }

Table A.140. Generic Representation of the Constraint of the Constraint Type *Membership in Controlled Vocabularies*

context class	classes	c. element
Computer-Science-Book-Subject	Computer-Science-Book-Subjects, Computer-Science-Book-Topics, Computer-Science-Book-Categories	membership in controlled vocabularies

Table A.141. Generic Representation of the Constraint Type *Membership in Controlled Vocabularies*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	-	-	$<i_i>$ $(1 \leqslant i \leqslant n)$	membership in controlled vocabularies	-

A.69 Disjoint Properties

A constraint of the constraint type *disjoint properties (R-10)* states that all of the object or data properties p_i, 1 ⩽ i ⩽ n, are pairwise disjoint; that is, no individual i_1 can be connected to an individual i_2 or a literal l by both p_i and p_j for i ≠ j. The properties *author* and *title*, e.g., may be defined to be disjoint.

$\mathcal{K}$ = { author ⊑ ¬ title }

Table A.142. Generic Representation of the Constraint of the Constraint Type *Disjoint Properties*

context class	left p. list	right p. list	classes	c. element	c. value
⊤	author, title	-	-	disjoint properties	-

Table A.143. Generic Representation of the Constraint Type *Disjoint Properties*

context class	left p. list	right p. list	classes	c. element	c. value
⊤	$<p_i>$ $(1 \leqslant i \leqslant n)$	-	-	disjoint properties	-

A.70 Disjoint Classes

The constraint type *disjoint classes (R-7)* states that all of the classes C_i, 1 ⩽ i ⩽ n, are pairwise disjoint; that is, no individual can be at the same time an instance of both C_i and C_j for i ≠ j. Nothing can be a book and a journal article at the same time is an example of a constraint of this type.

$\mathcal{K}$ = { Book ⊑ ¬ Journal-Article }

Table A.144. Generic Representation of the Constraint of the Constraint Type *Disjoint Classes*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	-	-	Book, Journal-Article	disjoint classes	-

Table A.145. Generic Representation of the Constraint Type *Disjoint Classes*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	-	-	$<C_i>$ $(1 \leqslant i \leqslant n)$	disjoint classes	-

A.71 String Operations

Some constraints require building new strings out of existing strings. Restricting the string length of literals of certain data properties within the context of certain classes would be another constraint of the constraint type *string operations (R-194)*. This constraint type enables to provide an implementation for each SPARQL string function. Constraints of this type are checked for each instance of the class $C_{context}$ and take the properties p_i $(1 \leqslant i \leqslant n)$ and the constraining value as arguments for individual SPARQL string functions. There are cases, e.g., where the length of human-readable labels of studies (*rdfs:label*) like "2005" must exactly be 4.

Table A.146. Generic Representation of the Constraint of the Constraint Type *String Operations*

context class	left p. list	right p. list	classes	c. element	c. value
disco:Study	rdfs:label	-	-	STRLEN	4

Table A.147. Generic Representation of the Constraint Type *String Operations*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p_i>$ $(1 \leqslant i \leqslant n)$	-	-	<SPARQL string function>	<argument(s)>

A.72 Aggregations

Some constraints require aggregating multiple values of an object or data property p within the context of a class $C_{context}$, especially using the SPARQL aggregation set functions Count, Min, Max, Sum, and Avg or equivalents in other languages. (1) The number of codes of a given variable must be below a maximum value, (2) the sum of percentages of all codes of a given variable must exactly be 100, (3) the absolute frequency of all valid codes of a given variable must be equal to a given value, or (4) the number of questions of a given questionnaire must be exactly the value 20 are example constraints of the constraint type *aggregations (R-233)*.

Table A.148. Generic Representation of the Constraint of the Constraint Type *Aggregations*

context class	left p. list	right p. list	classes	c. element	c. value
disco:Questionnaire	disco:question	-	-	count	20

Table A.149. Generic Representation of the Constraint Type *Aggregations*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p>$	-	-	<aggregation function>	<argument>

A.73 Individual Equality

Individual equality in DL indicates that two different names are known to refer to the same individual [27]. A constraint of the constraint type *individual equality (R-6)* states that all of the individuals i_i $(1 \leqslant i \leqslant n)$ are equal to each other which allows one to use each i_i as a synonym for each i_j — that is, in any constraint, i_i can be replaced with i_j without affecting the meaning of the constraint. The English book *The-Lord-Of-The-Rings*, e.g., should be handled as a synonym of the German book *Der-Herr-Der-Ringe*.

$$\mathcal{K} = \{\ \{\texttt{The-Lord-Of-The-Rings}\} = \{\texttt{Der-Herr-Der-Ringe}\}\ \}$$

Table A.150. Generic Representation of the Constraint of the Constraint Type *Individual Equality*

context class	left p. list	right p. list	classes	c. element	c. value
The-Lord-Of-The-Rings, Der-Herr-Der-Ringe	-	-	-	equal individuals	-

Table A.151. Generic Representation of the Constraint Type *Individual Equality*

context class	left p. list	right p. list	classes	c. element	c. value
$<i_i>$ $(1 \leqslant i \leqslant n)$	-	-	-	equal individuals	-

A.74 Individual Inequality

A constraint of the constraint type *individual inequality (R-14)* states that all of the individuals i_i $(1 \leqslant i \leqslant n)$ are different from each other; that is, no individuals i_i and i_j with i ≠ j can be derived to be equal which may be the case in an nUNA setting. With the next constraint of this type, it can be formalized that the books *The-Adventures-Of-Sherlock-Holmes* and *The-Memoirs-Of-Sherlock-Holmes* are actually different.

$$\mathcal{K} = \{\ \{\texttt{The-Adventures-Of-Sherlock-Holmes}\} \neq \{\texttt{The-Memoirs-Of-Sherlock-Holmes}\}\ \}$$

Table A.152. Generic Representation of the Constraint of the Constraint Type *Individual Inequality*

context class	left p. list	right p. list	classes	c. element	c. value
The-Adventures-Of-Sherlock-Holmes, The-Memoirs-Of-Sherlock-Holmes	-	-	-	different individuals	-

Table A.153. Generic Representation of the Constraint Type *Individual Inequality*

context class	left p. list	right p. list	classes	c. element	c. value
$<i_i>$ $(1 \leqslant i \leqslant n)$	-	-	-	different individuals	-

A.75 Context-Specific Valid Classes

With constraints of the constraint type *context-specific valid classes (R-209)*, one can state which classes C_i $(1 \leqslant i \leqslant n)$ of resources are valid to be used within a certain context. The context can be a graph, an application profile, an input stream, a data creation function, or an API. For future versions of RDF vocabularies, out-dated classes can indirectly be marked as deprecated. Within the context of the currently validated application profile, e.g., it should only be allowed to use the classes *Book*, *Journal*, and *Proceedings*.

Table A.154. Generic Representation of the Constraint of the Constraint Type *Context-Specific Valid Classes*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	-	-	Book, Journal, Proceedings	context-specific valid classes	-

Table A.155. Generic Representation of the Constraint Type *Context-Specific Valid Classes*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	-	-	$<C_i>$ $(1 \leqslant i \leqslant n)$	context-specific valid classes	-

A.76 Context-Specific Valid Properties

With constraints of the constraint type *context-specific valid properties (R-210)*, one can state which properties p_i $(1 \leqslant i \leqslant n)$ are valid to be used within a certain context. The context can be a graph, an application profile, a data receipt function, a data creation function, or an API. For future versions of RDF vocabularies, out-dated properties can indirectly be marked as deprecated. Within the context of the currently validated application profile, e.g., it should only be allowed to use the properties *author*, *subject*, and *title*.

Table A.156. Generic Representation of the Constraint of the Constraint Type *Context-Specific Valid Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	author, subject, title	-	-	context-specific valid properties	-

Table A.157. Generic Representation of the Constraint Type *Context-Specific Valid Properties*

context class	left p. list	right p. list	classes	c. element	c. value
$\top$	$<p_i>$ $(1 \leqslant i \leqslant n)$	-	-	context-specific valid properties	-

A.77 Property Assertions

The constraint type *property assertions (R-96)* includes positive property assertions and negative property assertions. A positive object property assertion states that the individual i_1 is connected by the object property p to the individual i_2. A negative object property assertion states that the individual i_1 is not connected by the object property p to the individual i_2. A positive data property assertion states that the individual i_1 is connected by the data property p to the literal l. A negative data property assertion states that the individual i_1 is not connected by the data property p to the literal l. The author of the book *The-Hunger-Games*, e.g., must be *Suzanne-Collins*.

$$\mathcal{K} = \{ \texttt{author(The-Hunger-Games, Suzanne-Collins)} \}$$

Table A.158. Generic Representation of the Constraint of the Constraint Type *Property Assertions*

context class	left p. list	right p. list	classes	c. element	c. value
The-Hunger-Games	author	-	Suzanne-Collins	property assertion	-

Table A.159. Generic Representation of the Constraint Type *Property Assertions*

context class	left p. list	right p. list	classes	c. element	c. value
$<i_1>$	$<p>$		$<i_2>$ | $<l>$	property assertion	-

A.78 Intersection

DLs allow new concepts and roles to be built using a variety of different constructors. We distinguish concept and role constructors depending on whether concept or role expressions are constructed. In the case of concepts, one can further separate basic boolean constructors, role restrictions and nominals/enumerations. Boolean concept constructors provide basic boolean operations that are closely related to the familiar operations of intersection, union, and complement of sets, or to conjunction, disjunction, and negation of logical expressions [27].

Concept inclusions allow us to state that all mothers are female and that all mothers are parents, but what we really mean is that mothers are exactly the female parents. DLs support such statements by allowing us to form complex concepts such as the *intersection (R-15/16)* (also called *composition* or *conjunction*) which denotes the set of individuals that are both female and

parents. A complex concept can be used in exactly the same way as an atomic concept, e.g., in the equivalence Mother ≡ Female ⊓ Parent. When using this intersection in terms of a constraint, a constraint violation is raised if a mother is not at the same time female and a parent.

A constraint of the constraint type *intersection* defines either (1) a class $C_{context}$ which contains all individuals that are instances of all classes C_i for $1 \leqslant \mathrm{i} \leqslant \mathrm{n}$ or (2) a datatype $DT_{context}$ which includes all literals that are contained in each datatype DT_i for $1 \leqslant \mathrm{i} \leqslant \mathrm{n}$.

$$\mathcal{K} = \{\ \texttt{Mother} \equiv \texttt{Female} \sqcap \texttt{Parent}\ \}$$

Table A.160. Generic Representation of the Constraint of the Constraint Type *Intersection*

context class	left p. list	right p. list	classes	c. element	c. value
Mother	-	-	Female, Parent	intersection	-

Table A.161. Generic Representation of the Constraint Type *Intersection*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$ | $<DT_{context}>$	-	-	$<C_i>$ $(1 \leqslant i \leqslant n)$ | $<DT_i>$ $(1 \leqslant i \leqslant n)$	intersection	-

A.79 Disjunction

Synonyms for *disjunction (R-17/18)* of classes or datatypes are *union* and *inclusive or*. A union of classes $C_{context}$ contains all individuals that are instances of at least one of these classes C_i for $1 \leqslant \mathrm{i} \leqslant \mathrm{n}$. A union of datatypes $DT_{context}$ contains all literals that are instances of at least one of these datatypes DT_i for $1 \leqslant \mathrm{i} \leqslant \mathrm{n}$. A publication, e.g., is either a book or a journal article or an article in conference proceedings or an article in workshop proceedings or a technical report.

$$\mathcal{K} = \{$$
$$\texttt{Publication} \equiv \texttt{Book} \sqcup \texttt{Journal-Article} \sqcup \texttt{Article-In-Conference-Proceedings} \sqcup \}$$
$$\texttt{Article-In-Workshop-Proceedings} \sqcup \texttt{Technical-Report}\ \}$$

Table A.162. Generic Representation of the Constraint of the Constraint Type *Disjunction*

context class	left p. list	right p. list	classes	c. element	c. value
Publication	-	-	Book, Journal-Article, ...	disjunction	-

Table A.163. Generic Representation of the Constraint Type *Disjunction*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$ | $<DT_{context}>$	-	-	$<C_i>$ $(1 \leqslant i \leqslant n)$ | $<DT_i>$ $(1 \leqslant i \leqslant n)$	disjunction	-

A.80 Negation

With constraints of the constraint type *negation (R-19/20)*, which corresponds to *complement* of classes or data ranges in DL terminology, it can be stated that (1) instances of class C_1 do not have to be instances of class C_2 or (2) literals of datatype DT_1 do not have to be of the datatype DT_2. Individuals of the type *Publication-Not-Book*, e.g., must only be publications that are not books.

$$\mathcal{K} = \{\ \texttt{Publication-Not-Book} \equiv \neg\texttt{Book}\ \}$$

Table A.164. Generic Representation of the Constraint of the Constraint Type *Negation*

context class	left p. list	right p. list	classes	c. element	c. value
Publication-Not-Book	-	-	Book	negation	-

Table A.165. Generic Representation of the Constraint Type *Negation*

context class		left p. list	right p. list	classes		c. element	c. value
$<C_1>$	$<DT_1>$	-	-	$<C_2>$	$<DT_2>$	negation	-

A.81 Default Values

Default values for objects (*R-31*) or literals (*R-38*) of given properties p within the context of given classes $C_{context}$ are inferred automatically when these properties are not present in the data. It should be possible to declare the default value for a given property, so that input forms can be pre-populated and to insert a required property that is missing in a web service call. A default value is normally used when creating resources. A service should use the default value to provide a value for a property if none is provided in the creation request. A default value may be used by consumers of a described resource if the defined property is not present in the representation of the described resource. Per default, the status of a book should be marked as published, i.e., the value of the property *isPublished* should be *true* for books in case the property is not stated.

Table A.166. Generic Representation of the Constraint of the Constraint Type *Default Values*

context class	left p. list	right p. list	classes	c. element	c. value
Book	isPublished	-	-	default value	true

Table A.167. Generic Representation of the Constraint Type *Default Values*

context class	left p. list	right p. list	classes	c. element	c. value
$<C_{context}>$	$<p>$	-	-	default value	<default value>

B

Constraint Type Specific Expressivity of Constraint Languages

We evaluated to which extend the most common constraint languages fulfill each of the 81 requirements to formulate constraints and to validate RDF data. We also evaluated to which extend a specific requirement is satisfied by the better performing OWL 2 profile *OWL 2 QL* or if the more expressive OWL 2 sub-language *OWL 2 DL* is needed. In case a constraint type is marked with an asterisk, reasoning may be performed prior to the validation of RDF data on constraints of that particular type to enhance data quality.

In case a constraint language does not directly support a given constraint type, but nevertheless is capable to express that particular constraint type, either by limitations, workarounds, or extensions, we mark that combination of constraint type and constraint language with the tilde symbol and consider the constraint type as being covered by the language.

For the constraint type *negative literal pattern matching (R-230)*, e.g., there is no dedicated term in the SHACL vocabulary. Nevertheless, this constraint type is expressible with SHACL in terms of a workaround by combining multiple language constructs. By combining the negation constraint (*sh:NotConstraint*) and the property *sh:pattern*, e.g., it can be ensured that an ISSN of a journal does not have be conform to a valid 10- or 13-digit ISBN. The property sh:pattern is used to validate whether all values of a given property match a given regular expression. The values of sh:pattern must be valid pattern arguments for the SPARQL REGEX function:

```
SHACL Shapes Graph:
:JournalShape
    a sh:Shape ;
    sh:constraint [
        a sh:NotConstraint ;
        sh:shape [
            a sh:Shape ;
            sh:property [
                sh:predicate :issn ;
                sh:pattern "(ISBN[-]*(1[03])*[ ]*(: )0,1)*(([0-9Xx][- ]*)
                           13|([0-9Xx][- ]*)10)" ; ] ; ] ] .

Valid journal:
```

```
:Journal-Of-Web-Semantics
    :issn "1570-8268" .

Invalid journal:
:International-Journal-Of-Metadata-Semantics-And-Ontologies
    :issn "978-3-642-35172-3" .
```

Table B.1. Constraint Type Specific Expressivity of Constraint Languages (1)

Constraint Types	DSP	OWL2-DL	OWL2-QL	ReSh	SHACL	ShEx	SPIN
*Functional Properties	✗	✓	✗	✗	✗	✗	✓
*Inverse-Functional Properties	✗	✓	✗	✗	✗	✗	✓
*Primary Key Properties	✗	✓	✗	✗	✗	✗	✓
*Subsumption	✗	✓	✓	~	✓	✓	✓
*Sub-Properties	✗	✓	✓	✗	✗	✗	✓
*Object Property Paths	✗	✓	✗	✗	✗	✗	✓
Allowed Values	✓	✓	✗	✓	✓	✓	✓
Not Allowed Values	✗	✓	✗	✗	~	✓	✓
*Class Equivalence	✗	✓	✓	✗	✗	✗	✓
*Equivalent Properties	✗	✓	✓	✗	✓	✗	✓
Literal Value Comparison	✗	✗	✗	✗	✓	✓	✓
Value is Valid for Datatype	✗	✗	✗	✗	✓	✗	✓
*Property Domains	✗	✓	✓	✗	✗	✗	✓
*Property Ranges	✗	✓	✓	✗	✓	✗	✓
*Class-Specific Property Range	✓	✓	✗	✓	✓	✓	✓
Data Property Facets	✗	✓	✓	✗	✓	✗	✓
Literal Ranges	✗	✓	✗	✗	✓	✗	✓
Negative Literal Ranges	✗	✓	✗	✗	~	✗	✓
IRI Pattern Matching	✗	✗	✗	✗	✗	✓	✓
Literal Pattern Matching	✗	✓	✗	✗	✓	✗	✓

Table B.2. Constraint Type Specific Expressivity of Constraint Languages (2)

Constraint Types	DSP	OWL2-DL	OWL2-QL	ReSh	SHACL	ShEx	SPIN
Negative Literal Pattern Matching	✗	✓	✗	✗	~	✗	✓
*Existential Quantifications	✗	✓	✗	~	✓	~	✓
*Universal Quantifications	✗	✓	✗	✗	✓	✗	✓
*Value Restrictions	✓	✓	✗	✓	✓	✓	✓
Use Sub-Super Relations in Validation	✗	✗	✗	✗	✗	✗	✓
Negative Property Constraints	✗	✓	✗	✗	~	✓	✓
Language Tag Matching	✗	✗	✗	✗	✗	✗	✓
Language Tag Cardinality	✗	✗	✗	✗	✗	✗	✓
Whitespace Handling	✗	✗	✗	✗	✗	✗	✓
HTML Handling	✗	✗	✗	✗	✗	✗	✓
Structure	✗	✗	✗	✗	✗	✗	✓
*Minimum Unqualified Cardinality	✓	✓	✗	~	✓	✓	✓
*Minimum Qualified Cardinality	✓	✓	✗	~	✓	✓	✓
*Maximum Unqualified Cardinality	✓	✓	✗	~	✓	✓	✓
*Maximum Qualified Cardinality	✓	✓	✗	~	✓	✓	✓
*Exact Unqualified Cardinality	✓	✓	✗	~	✓	✓	✓
*Exact Qualified Cardinality	✓	✓	✗	~	✓	✓	✓
*Cardinality Shortcuts	✗	✓	✗	✓	✗	✓	✓
Vocabulary	✗	✗	✗	✗	✗	✗	✓
Provenance	✗	✗	✗	✗	✗	✗	✓

Table B.3. Constraint Type Specific Expressivity of Constraint Languages (3)

Constraint Types	DSP	OWL2-DL	OWL2-QL	ReSh	SHACL	ShEx	SPIN
Required Properties	✓	✓	✗	✓	~	✓	✓
Optional Properties	✓	✓	✗	✓	~	✓	✓
Repeatable Properties	✓	✓	✗	✓	~	✓	✓
Conditional Properties	✗	✗	✗	✗	✓	✗	✓
Recommended Properties	✗	✗	✗	✗	✗	✗	✓
Severity Levels	✗	✗	✗	✗	✓	✗	✓
Labeling and Documentation	✗	✗	✗	✗	✓	✗	✓
Context-Sp. Property Groups	✗	~	~	✓	✓	✓	✓
Context-Sp. Exclusive OR of P.	✗	✓	✗	✗	✗	✓	✓
Context-Sp. Exclusive OR of P. Groups	✗	~	✗	✓	✗	✓	✓
Context-Sp. Inclusive OR of P.	✗	~	~	✗	✓	✗	✓
Context-Sp. Inclusive OR of P. Groups	✗	~	~	✗	✓	✗	✓
Mathematical Operations	✗	✗	✗	✗	✗	✗	✓
Ordering	✗	✗	✗	✗	✗	✗	✓
*Inverse Object Properties	✗	✓	✓	~	✓	✗	✓
*Symmetric Object Properties	✗	✓	✓	✗	✗	✗	✓
*Asymmetric Object Properties	✗	✓	✓	✗	✗	✗	✓
*Transitive Object Properties	✗	✓	✗	✗	✗	✗	✓
*Self Restrictions	✗	✓	✗	✗	✗	✗	✓
Valid Identifiers	✗	✗	✗	✗	✗	✗	✓

Table B.4. Constraint Type Specific Expressivity of Constraint Languages (4)

Constraint Types	**DSP**	**OWL2-DL**	**OWL2-QL**	**ReSh**	**SHACL**	**ShEx**	**SPIN**
Recursive Queries	✓	✓	✓	✓	✓	✓	~
*Reflexive Object Properties	✗	✓	✓	✗	✗	✗	✓
*Class-Sp. Reflexive Object P.	✗	✓	✗	✗	✗	✗	✓
*Irreflexive Object Properties	✗	✓	✓	✗	✗	✗	✓
*Class-Specific Irreflexive Object Properties	✗	✓	✗	✗	✗	✗	✓
Data Model Consistency	✗	✗	✗	✗	✗	✗	✓
Handle RDF Collections	✗	✗	✗	✗	✗	✗	✓
Membership in Controlled Vocabularies	✓	✗	✗	✗	✗	✗	✓
Disjoint Properties	✗	✓	✓	✗	✓	✗	✓
Disjoint Classes	✗	✓	✓	✗	✗	✗	✓
String Operations	✗	✗	✗	✗	~	✗	✓
Aggregations	✗	✗	✗	✗	✗	✗	✓
*Individual Equality	✗	✓	✗	✗	✗	✗	✓
Individual Inequality	✗	✓	✓	✗	✗	✗	✓
Context-Specific Valid Classes	✗	✗	✗	✗	✗	✗	✓
Context-Specific Valid Properties	✗	✗	✗	✗	✓	✗	✓
Property Assertions	✗	✓	~	✗	~	✗	✓
*Intersection	✗	✓	✗	✓	✓	✓	✓
*Disjunction	✗	✓	✗	✗	✓	✗	✓
*Negation	✗	✓	✗	✗	✓	✗	✓
*Default Values	✗	✗	✗	✓	✓	✗	✓

C

Classification of Constraints according to the RDF Constraints Vocabulary

Constraints on RDF data are either simple constraints or complex constraints. *Simple constraints* denotes the set of atomic constraints with respect to a single constraining element. In contrast, there are *complex constraints*, i.e., the set of constraints which are created out of simple and/or other complex constraints. *DL* denotes the set of constraints which are expressible in *Description Logics (DL)*.

Table C.1. Classification of Constraints according to the RDF Constraints Vocabulary (1)

Constraint Types	**Simple**	**DL**
Functional Properties	✓	✓
Inverse-Functional Properties	✗	✓
Primary Key Properties	~	✓
Subsumption	✓	✓
Sub-Properties	✓	✓
Object Property Paths	✓	✓
Allowed Values	✓	✓
Not Allowed Values	✗	✓
Class Equivalence	✗	✓
Equivalent Properties	✗	✓
Literal Value Comparison	✓	✗
Value is Valid for Datatype	✓	✗
Property Domains	~	✓
Property Ranges	~	✓
Class-Specific Property Range	~	✓
Data Property Facets	✓	✗
Literal Ranges	✗	✗
Negative Literal Ranges	✗	✗
IRI Pattern Matching	✓	✗
Literal Pattern Matching	✓	✗
Negative Literal Pattern Matching	✗	✗

Table C.2. Classification of Constraints according to the RDF Constraints Vocabulary (2)

Constraint Types	**Simple**	**DL**
Existential Quantifications	✓	✓
Universal Quantifications	✓	✓
Value Restrictions	✓	✓
Use Sub-Super Relations in Validation	✓	✗
Negative Property Constraints	✗	✓
Language Tag Matching	✓	✗
Language Tag Cardinality	✓	✗
Whitespace Handling	✓	✗
HTML Handling	✓	✗
Structure	✗	✗
Minimum Unqualified Cardinality	✓	✓
Minimum Qualified Cardinality	✓	✓
Maximum Unqualified Cardinality	✓	✓
Maximum Qualified Cardinality	✓	✓
Exact Unqualified Cardinality	~	✓
Exact Qualified Cardinality	~	✓
Cardinality Shortcuts	✓	✓
Vocabulary	✓	✗
Provenance	✓	✗
Required Properties	✓	✓
Optional Properties	~	✓
Repeatable Properties	✓	✓
Conditional Properties	✓	✓
Recommended Properties	✓	✗
Severity Levels	✓	✗
Labeling and Documentation	✗	✗
Context-Sp. Property Groups	✗	✓
Context-Sp. Exclusive OR of P.	✗	✓
Context-Sp. Exclusive OR of P. Groups	✗	✓
Context-Sp. Inclusive OR of P.	✗	✓

Table C.3. Classification of Constraints according to the RDF Constraints Vocabulary (3)

Constraint Types	**Simple**	**DL**
Context-Sp. Inclusive OR of P. Groups	✗	✓
Mathematical Operations	✓	✗
Ordering	✓	✗
Inverse Object Properties	✓	✓
Symmetric Object Properties	✓	✓
Asymmetric Object Properties	~	✓
Transitive Object Properties	✓	✓
Self Restrictions	✗	✓
Valid Identifiers	✓	✗
Recursive Queries	✓	✓
Reflexive Object Properties	~	✓
Class-Sp. Reflexive Object P.	✓	✓
Irreflexive Object Properties	~	✓
Class-Specific Irreflexive Object Properties	~	✓
Data Model Consistency	✗	✗
Handle RDF Collections	✓	✗
Membership in Controlled Vocabularies	✗	✓
Disjoint Properties	✓	✓
Disjoint Classes	✗	✓
String Operations	✓	✗
Aggregations	✓	✗
Individual Equality	✗	✓
Individual Inequality	✗	✓
Context-Specific Valid Classes	✓	✗
Context-Specific Valid Properties	✓	✗
Property Assertions	✓	✓
Intersection	✓	✓
Disjunction	✓	✓
Negation	✓	✓
Default Values	✓	✗

D

CWA and UNA Dependency of Constraint Types

Validation and reasoning assume different semantics which may lead to different validation results when applied to particular constraint types. Reasoning requires the *open-world assumption* (*OWA*) with the *non-unique name assumption* (*nUNA*), whereas validation is classically based on the *closed-world assumption* (*CWA*) and the *unique name assumption* (*UNA*). Therefore, we investigate for each constraint type if validation results differ (1) if the CWA or the OWA and (2) if the UNA or the nUNA is assumed, i.e., we examine for each constraint type (1) if it depends on the CWA and (2) if it depends on the UNA.

Table D.1. CWA and UNA Dependency of Constraint Types (1)

	Dependency	
Constraint Types	*CWA*	*UNA*
Functional Properties	✓	✓
Inverse-Functional Properties	✓	✓
Primary Key Properties	✓	✓
Subsumption	✓	✓
Sub-Properties	✓	✓
Object Property Paths	✓	✓
Allowed Values	✗	✓
Not Allowed Values	✗	✓
Class Equivalence	✓	✓
Equivalent Properties	✓	✓
Literal Value Comparison	✗	✗
Value is Valid for Datatype	✗	✗
Property Domains	✓	✓
Property Ranges	✓	✓

Table D.2. CWA and UNA Dependency of Constraint Types (2)

	Dependency	
Constraint Types	*CWA*	*UNA*
Class-Specific Property Range	✓	✓
Data Property Facets	✗	✗
Literal Ranges	✗	✗
Negative Literal Ranges	✗	✗
IRI Pattern Matching	✗	✓
Literal Pattern Matching	✗	✗
Negative Literal Pattern Matching	✗	✗
Existential Quantifications	✓	✓
Universal Quantifications	✗	✓
Value Restrictions	✓	✓
Use Sub-Super Relations in Validation	✗	✓
Negative Property Constraints	✗	✓
Language Tag Matching	✓	✗
Language Tag Cardinality	✓	✓
Whitespace Handling	✗	✗
HTML Handling	✗	✗
Structure	✓	✓
Minimum Unqualified Cardinality	✓	✓
Minimum Qualified Cardinality	✓	✓
Maximum Unqualified Cardinality	✗	✓
Maximum Qualified Cardinality	✗	✓
Exact Unqualified Cardinality	✓	✓
Exact Qualified Cardinality	✓	✓
Cardinality Shortcuts	✓	✓
Vocabulary	✓	✓
Provenance	✓	✓
Required Properties	✓	✓
Optional Properties	✗	✗
Repeatable Properties	✓	✗
Conditional Properties	✓	✓
Recommended Properties	✓	✓
Severity Levels	✓	✗
Labeling and Documentation	✓	✓

Table D.3. CWA and UNA Dependency of Constraint Types (3)

	Dependency	
Constraint Types	*CWA*	*UNA*
Context-Sp. Property Groups	✓	✓
Context-Sp. Exclusive OR of P.	✗	✓
Context-Sp. Exclusive OR of P. Groups	✗	✓
Context-Sp. Inclusive OR of P.	✓	✓
Context-Sp. Inclusive OR of P. Groups	✓	✓
Mathematical Operations	✗	✗
Ordering	✓	✗
Inverse Object Properties	✓	✓
Symmetric Object Properties	✓	✓
Asymmetric Object Properties	✗	✗
Transitive Object Properties	✓	✓
Self Restrictions	✓	✓
Valid Identifiers	✗	✓
Recursive Queries	✗	✗
Reflexive Object Properties	✓	✓
Class-Sp. Reflexive Object P.	✓	✓
Irreflexive Object Properties	✗	✗
Class-Specific Irreflexive Object Properties	✗	✓
Data Model Consistency	✓	✓
Handle RDF Collections	✗	✗
Membership in Controlled Vocabularies	✓	✓
Disjoint Properties	✗	✗
Disjoint Classes	✗	✓
String Operations	✗	✗
Aggregations	✗	✗
Individual Equality	✓	✗
Individual Inequality	✗	✗
Context-Specific Valid Classes	✗	✗
Context-Specific Valid Properties	✗	✗
Property Assertions	✓	✓
Intersection	✓	✓
Disjunction	✓	✓
Negation	✗	✗
Default Values	✓	✓

E

Constraining Elements for Constraint Types

The simple structure of its conceptual model using a small lightweight vocabulary plus the constraining elements form the building blocks of our proposed validation framework for RDF-based constraint languages. In this chapter, we list for every constraint type its representation in our framework which not only shows that constraints of any constraint type can indeed be described generically in this way, but which also forms the starting point for any mapping to SPIN using this framework.

Simple constraints denotes the set of atomic constraints with respect to a single constraining element. Constraining elements are, e.g., taken from DL. Another example of a constraining element is the SPARQL function REGEX where a regular expression is checked against some property value. Complex constraints, i.e., the set of constraints which are created out of simple and/or other complex constraints, need several constraining elements to be expressed. The *constraining element* is an intuitive term which indicates the actual type of constraint. Constraining elements are closer to atomic elements of constraints.

In most cases, a constraining element directly corresponds to a single constraint type, sometimes it is shared by several constraint types, and in a few cases only the interplay of multiple constraining elements ensures that each possible constraint of a certain type can be expressed. For only three constraints types, the constraining element *SPARQL* indicates that constraints of these types may be of such complexity that only plain SPARQL can be used for their proper formulation.

If constraint types are expressible in DL, constraining elements are formally based on DL constructs like concept and role constructors ($\sqsubseteq$, $\equiv$, $\sqcap$, $\sqcup$, $\neg$, $\exists$, $\forall$, $\geqslant$, $\leqslant$), equality ($=$), and inequality ($\neq$). In case constraint types cannot be expressed in DL, we reuse widely known terms from SPARQL (e.g., REGEX) or XML Schema constraining facets (e.g., *xsd:minInclusive*) as constraining elements. In this chapter, we also list for each in DL expressible constraint type the DL constructs needed to formulate constraints of that particular type.

Table E.1. Expression of Constraint Types in DL (1)

Constraint Types	**Expression in DL**
Functional Properties	functional
Inverse-Functional Properties	inverse, functional
Primary Key Properties	inverse, functional
Subsumption	$\sqsubseteq$
Sub-Properties	$\sqsubseteq$
Object Property Paths	$\sqsubseteq$
Allowed Values	$\sqcup$
Not Allowed Values	$\sqcup$, $\neg$
Class Equivalence	$\equiv$
Equivalent Properties	$\equiv$
Literal Value Comparison	$>$ or $\geqslant$ or $<$ or $\leqslant$ or $=$ or $\neq$
Value is Valid for Datatype	-
Property Domains	$\exists$, $\sqsubseteq$
Property Ranges	$\sqsubseteq$, $\forall$
Class-Specific Property Range	$\sqsubseteq$, $\exists$, $\neg$
Data Property Facets	-
Literal Ranges	-
Negative Literal Ranges	-
IRI Pattern Matching	-
Literal Pattern Matching	-
Negative Literal Pattern Matching	-
Existential Quantification	$\exists$
Universal Quantification	$\forall$
Value Restrictions	$\exists$
Use Sub-Super Relations in Validation	-
Negative Property Constraints	$\neg$, $\exists$
Language Tag Matching	-
Language Tag Cardinality	-
Whitespace Handling	-
HTML Handling	-
Structure	-
Minimum Unqualified Cardinality	$\geqslant$
Minimum Qualified Cardinality	$\geqslant$
Maximum Unqualified Cardinality	$\leqslant$
Maximum Qualified Cardinality	$\leqslant$
Exact Unqualified Cardinality	($\geqslant$, $\leqslant$, $\sqcap$) or $=$
Exact Qualified Cardinality	($\geqslant$, $\leqslant$, $\sqcap$) or $=$
Cardinality Shortcuts	-
Vocabulary	-
Provenance	-
Required Properties	$\exists$

Table E.2. Expression of Constraint Types in DL (2)

Constraint Types	**Expression in DL**
Optional Properties	$\exists$, $\sqsubseteq$
Repeatable Properties	$\geqslant$
Conditional Properties	$\sqsubseteq$
Recommended Properties	-
Severity Levels	-
Labeling and Documentation	-
Context-Sp. Property Groups	$\sqcap$
Context-Sp. Exclusive OR of P.	$\neg$, $\sqcap$, $\sqcup$
Context-Sp. Exclusive OR of P. Groups	$\neg$, $\sqcap$, $\sqcup$
Context-Sp. Inclusive OR of P.	$\sqcup$
Context-Sp. Inclusive OR of P. Groups	$\sqcup$, $\sqcap$
Mathematical Operations	-
Ordering	-
Inverse Object Properties	inverse
Symmetric Object Properties	symmetric
Asymmetric Object Properties	asymmetric
Transitive Object Properties	$\sqsubseteq$
Self Restrictions	$\exists$
Valid Identifiers	-
Recursive Queries	-
Reflexive Object Properties	reflexive
Class-Sp. Reflexive Object P.	reflexive
Irreflexive Object Properties	reflexive, $\neg$
Class-Specific Irreflexive Object Properties	reflexive, $\neg$
Data Model Consistency	-
Handle RDF Collections	-
Membership in Controlled Vocabularies	$\forall$, $\sqcap$, $\sqcup$
Disjoint Properties	$\sqsubseteq$, $\neg$
Disjoint Classes	$\sqcap$, $\sqsubseteq$
String Operations	-
Aggregations	-
Individual Equality	$=$
Individual Inequality	$\neq$
Context-Specific Valid Classes	-
Context-Specific Valid Properties	-
Property Assertions	$=$ or $\neq$
Intersection	$\sqcap$
Disjunction	$\sqcup$
Negation	$\neg$
Default Values	-

Table E.3. Constraining Elements for Constraint Types (1)

Constraint Types	**Constraining Elements**
Functional Properties	functional properties
Inverse-Functional Properties	inverse-functional properties
Primary Key Properties	primary key
Subsumption	sub-class
Sub-Properties	sub-property
Object Property Paths	object property path
Allowed Values	allowed values
Not Allowed Values	not allowed values
Class Equivalence	equivalent classes
Equivalent Properties	equivalent properties
Literal Value Comparison	mathematical symbols: is greater than, is greater than or equal to, is less than, is less than or equal to, is equal to, is not equal to
Value is Valid for Datatype	value is valid for datatype
Property Domains	property domain
Property Ranges	property range
Class-Specific Property Range	property range
Data Property Facets	SPARQL functions: REGEX, STRLEN XML Schema constraining facets: xsd:length, xsd:minLength, xsd:maxLength, xsd:enumeration, xsd:whiteSpace, xsd:maxInclusive, xsd:maxExclusive,xsd:minExclusive, xsd:minInclusive, xsd:pattern, xsd:totalDigits, xsd:fractionDigits
Literal Ranges	XML Schema constraining facets on values: xsd:minInclusive, xsd:maxExclusive xsd:maxInclusive, xsd:minExclusive
Negative Literal Ranges	not <XML Schema constraining facet on values> XML Schema constraining facets on values: xsd:minInclusive, xsd:maxExclusive xsd:maxInclusive, xsd:minExclusive
IRI Pattern Matching	IRI pattern matching
Literal Pattern Matching	REGEX, xsd:pattern
Negative Literal Pattern Matching	negative literal pattern matching
Existential Quantification	existential quantification
Universal Quantification	universal quantification
Value Restrictions	value restriction
Use Sub-Super Relations in Validation	use sub-super relations
Negative Property Constraints	negative properties
Language Tag Matching	language tag matching
Language Tag Cardinality	language tag minimum cardinality, language tag maximum cardinality, language tag exact cardinality
Whitespace Handling	whitespace handling
HTML Handling	html handling
Structure	SPARQL
Minimum Unqualified Cardinality	minimum cardinality
Minimum Qualified Cardinality	minimum cardinality
Maximum Unqualified Cardinality	maximum cardinality
Maximum Qualified Cardinality	maximum cardinality
Exact Unqualified Cardinality	exact cardinality
Exact Qualified Cardinality	exact cardinality

Table E.4. Constraining Elements for Constraint Types (2)

Constraint Types	**Constraining Elements**
Cardinality Shortcuts	pairs of cardinality shortcuts: optional and non-repeatable properties, optional and repeatable properties, mandatory and non-repeatable properties, mandatory and repeatable properties
Vocabulary	vocabulary
Provenance	provenance
Required Properties	required properties
Optional Properties	optional properties
Repeatable Properties	repeatable properties
Conditional Properties	conditional properties
Recommended Properties	recommended properties
Severity Levels	severity level
Labeling and Documentation	SPARQL
Context-Sp. Property Groups	property group
Context-Sp. Exclusive OR of P.	exclusive or
Context-Sp. Exclusive OR of P. Groups	exclusive or
Context-Sp. Inclusive OR of P.	inclusive or
Context-Sp. Inclusive OR of P. Groups	inclusive or
Mathematical Operations	Mathematical operations: addition, subtraction, multiplication, division
Ordering	ordered values
Inverse Object Properties	inverse property
Symmetric Object Properties	symmetric property
Asymmetric Object Properties	asymmetric property
Transitive Object Properties	transitive property
Self Restrictions	self restriction
Valid Identifiers	valid identifiers
Recursive Queries	recursive
Reflexive Object Properties	reflexive property
Class-Sp. Reflexive Object P.	reflexive property
Irreflexive Object Properties	irreflexive property
Class-Specific Irreflexive Object Properties	irreflexive property
Data Model Consistency	SPARQL
Handle RDF Collections	actions on RDF collections: add element to list, ...
Membership in Controlled Vocabularies	membership in controlled vocabularies
Disjoint Properties	disjoint properties
Disjoint Classes	disjoint classes
String Operations	SPARQL string functions: STRLEN, SUBSTR, UCASE, LCASE, STRSTARTS, STRENDS, CONTAINS, STRBEFORE, STRAFTER, ENCODE_FOR_URI, CONCAT, langMatches, REGEX, REPLACE
Aggregations	aggregation functions: count
Individual Equality	equal individuals
Individual Inequality	different individuals
Context-Specific Valid Classes	context-specific valid classes
Context-Specific Valid Properties	context-specific valid properties
Property Assertions	property assertion
Intersection	intersection
Disjunction	disjunction
Negation	negation
Default Values	default value

F

Software

XSD 2 OWL

Stand-alone application to automatically transform arbitrary XML Schemas into OWL ontologies based on formal logics and the XML Schema meta-model.

- Source code online available at:
  https://github.com/github-thomas-hartmann/phd-thesis

Validation Environment

Validation environment to (1) define constraints expressed in arbitrary RDF-based constraint languages and (2) validate RDF data according to these constraints.

- Online available at: http://purl.org/net/rdfval-demo
- Source code online available at:
  https://github.com/github-thomas-hartmann/phd-thesis

G

Publications

Main aspects of this thesis have been published in form of research papers. To make the research results of this thesis available in a sustainable way, all research results together with direct links to them can be found on a GitHub repository with the base URL https://github.com/github-thomas-hartmann/phd-thesis, so the complete set of publications. In this chapter, we list the references to these publications and group them by chapter (see Section G.1) and publication type (see Section G.2). Please note that in 2015, the last name of the author of this thesis changed from *Bosch* to *Hartmann*.

G.1 Publications by Chapter

Chapter 3:
Vocabularies for Representing Research Data and its Metadata

1. Block, W., **Bosch, Thomas**, Fitzpatrick, B., Gillman, D., Greenfield, J., Gregory, A., Hebing, M., Hoyle, L., Humphrey, C., Johnson, J., Linnerud, J., Mathiak, B., McEachern, S., Radler, B., Risnes, Ø., Smith, D., Thomas, W., Wackerow, J., Wegener, D., & Zenk-Möltgen, W. (2012). Developing a Model-Driven DDI Specification. *DDI Working Paper Series*
2. **Bosch, Thomas**, Cyganiak, R., Gregory, A., & Wackerow, J. (2013a). DDI-RDF Discovery Vocabulary: A Metadata Vocabulary for Documenting Research and Survey Data. In *Proceedings of the 6th Workshop on Linked Data on the Web (LDOW 2013), 22nd International World Wide Web Conference (WWW 2013)*, volume 996 Rio de Janeiro, Brazil. http://ceur-ws.org/Vol-996/
3. **Bosch, Thomas**, Cyganiak, R., Wackerow, J., & Zapilko, B. (2012). Leveraging the DDI Model for Linked Statistical Data in the Social, Behavioural, and Economic Sciences. In *Proceedings of the 12th DCMI International Conference on Dublin Core and Metadata Applications*

(DC 2012) Kuching, Sarawak, Malaysia. http://dcpapers.dublincore.org/pubs/article/view/3654

4. **Bosch, Thomas**, Cyganiak, R., Wackerow, J., & Zapilko, B. (2016). *DDI-RDF Discovery Vocabulary: A Vocabulary for Publishing Metadata about Data Sets (Research and Survey Data) into the Web of Linked Data*. DDI Alliance Specification, DDI Alliance. http://rdf-vocabulary.ddialliance.org/discovery
5. **Bosch, Thomas** & Mathiak, B. (2015). Use Cases Related to an Ontology of the Data Documentation Initiative. *IASSIST Quarterly*, 38(4) & 39(1), 25–37. http://iassistdata.org/iq/issue/38/4
6. **Bosch, Thomas**, Olsson, O., Gregory, A., & Wackerow, J. (2015c). DDI-RDF Discovery - A Discovery Model for Microdata. *IASSIST Quarterly*, 38(4) & 39(1), 17–24. http://iassistdata.org/iq/issue/38/4
7. **Bosch, Thomas**, Wira-Alam, A., & Mathiak, B. (2011). Designing an Ontology for the Data Documentation Initiative. In *Proceedings of the 8th Extended Semantic Web Conference (ESWC 2011), Poster-Session* Heraklion, Greece. http://www.eswc2011.org/content/accepted-posters.html
8. **Bosch, Thomas**, Wira-Alam, A., & Mathiak, B. (2014). Designing an Ontology for the Data Documentation Initiative. *Computing Research Repository (CoRR)*, abs/1402.3470. http://arxiv.org/abs/1402.3470
9. **Bosch, Thomas** & Zapilko, B. (2015). Semantic Web Applications for the Social Sciences. *IASSIST Quarterly*, 38(4) & 39(1), 7–16. http://iassistdata.org/iq/issue/38/4
10. **Bosch, Thomas**, Zapilko, B., Wackerow, J., & Gregory, A. (2013b). Towards the Discovery of Person-Level Data - Reuse of Vocabularies and Related Use Cases. In *Proceedings of the 1st International Workshop on Semantic Statistics (SemStats 2013), 12th International Semantic Web Conference (ISWC 2013),* Sydney, Australia. http://semstats.github.io/2013/proceedings
11. Schaible, J., Zapilko, B., **Bosch, Thomas**, & Zenk-Möltgen, W. (2015). Linking Study Descriptions to the Linked Open Data Cloud. *IASSIST Quarterly*, 38(4) & 39(1), 38–46. http://iassistdata.org/iq/issue/38/4
12. Vompras, J., Gregory, A., **Bosch, Thomas**, & Wackerow, J. (2015). Scenarios for the DDI-RDF Discovery Vocabulary. *DDI Working Paper Series*. http://dx.doi.org/10.3886/DDISemanticWeb02
13. Wackerow, J., Hoyle, L., & **Bosch, Thomas** (2016). *Physical Data Description*. DDI Alliance Specification, DDI Alliance. http://rdf-vocabulary.ddialliance.org/phdd.html

Chapter 4:
RDFication of XML Enabling to use RDF Validation Technologies

1. **Bosch, Thomas** (2012). Reusing XML Schemas' Information as a Foundation for Designing Domain Ontologies. In P. Cudré-Mauroux, J.

Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J. Parreira, J. Hendler, G. Schreiber, A. Bernstein, & E. Blomqvist (Eds.), *The Semantic Web - ISWC 2012*, volume 7650 of *Lecture Notes in Computer Science* (pp. 437–440). Springer Berlin Heidelberg. http://dx.doi.org/10.1007/978-3-642-35173-0_34

2. **Bosch, Thomas** & Mathiak, B. (2011). Generic Multilevel Approach Designing Domain Ontologies Based on XML Schemas. In *Proceedings of the 1st Workshop Ontologies Come of Age in the Semantic Web (OCAS 2011), 10th International Semantic Web Conference (ISWC 2011)* (pp. 1–12). Bonn, Germany. http://ceur-ws.org/Vol-809/
3. **Bosch, Thomas** & Mathiak, B. (2012). XSLT Transformation Generating OWL Ontologies Automatically Based on XML Schemas. In *Proceedings of the 6th International Conference for Internet Technology and Secured Transactions (ICITST 2011), IEEE Xplore Digital Library* (pp. 660–667). Abu Dhabi, United Arab Emirates. http://edas.info/web/icitst2011/program.html
4. **Bosch, Thomas** & Mathiak, B. (2013a). *Evaluation of a Generic Approach for Designing Domain Ontologies Based on XML Schemas*. Gesis Technical Report 08, Gesis - Leibniz Institute for the Social Sciences, Mannheim, Germany. http://www.gesis.org/publikationen/archiv/gesis-technical-reports/
5. **Bosch, Thomas** & Mathiak, B. (2013b). How to Accelerate the Process of Designing Domain Ontologies based on XML Schemas. *International Journal of Metadata, Semantics and Ontologies - Special Issue on Metadata, Semantics and Ontologies for Web Intelligence*, 8(3), 254 – 266. http://www.inderscience.com/info/inarticle.php?artid=57760

Chapter 5:
RDF Validation Requirements and Types of Constraints on RDF Data

1. Alonen, M., **Bosch, Thomas**, Charles, V., Clayphan, R., Coyle, K., Dröge, E., Isaac, A., Matienzo, M., Pohl, A., Rühle, S., & Svensson, L. (2015a). *Report on the Current State: Use Cases and Validation Requirements*. DCMI Draft, Dublin Core Metadata Initiative (DCMI). http://wiki.dublincore.org/index.php/RDF_Application_Profiles/UCR_Deliverable
2. Alonen, M., **Bosch, Thomas**, Charles, V., Clayphan, R., Coyle, K., Dröge, E., Isaac, A., Matienzo, M., Pohl, A., Rühle, S., & Svensson, L. (2015b). *Report on Validation Requirements*. DCMI Draft, Dublin Core Metadata Initiative (DCMI). http://wiki.dublincore.org/index.php/RDF_Application_Profiles/Requirements
3. **Bosch, Thomas** & Eckert, K. (2014a). Requirements on RDF Constraint Formulation and Validation. In *Proceedings of the 14th DCMI International Conference on Dublin Core and Metadata Applications (DC 2014)*

Austin, Texas, USA. http://dcevents.dublincore.org/IntConf/dc-2014/paper/view/257

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

Chapter 6:
Providing Consistent Implementations for any RDF-based Constraint Language

1. **Bosch, Thomas** & Eckert, K. (2014b). Towards Description Set Profiles for RDF using SPARQL as Intermediate Language. In *Proceedings of the 14th DCMI International Conference on Dublin Core and Metadata Applications (DC 2014)* Austin, Texas, USA. http://dcevents.dublincore.org/IntConf/dc-2014/paper/view/270

Chapter 7:
Validation Framework for RDF-based Constraint Languages

1. **Bosch, Thomas** & Eckert, K. (2015). Guidance, Please! Towards a Framework for RDF-based Constraint Languages. In *Proceedings of the 15th DCMI International Conference on Dublin Core and Metadata Applications (DC 2015)* São Paulo, Brazil. http://dcevents.dublincore.org/IntConf/dc-2015/paper/view/386/368
2. **Bosch, Thomas**, Nolle, A., Acar, E., & Eckert, K. (2015b). RDF Validation Requirements - Evaluation and Logical Underpinning. *Computing Research Repository (CoRR)*, abs/1501.03933. http://arxiv.org/abs/1501.03933

Chapter 8:
The Role of Reasoning for RDF Validation

1. **Bosch, Thomas**, Acar, E., Nolle, A., & Eckert, K. (2015a). The Role of Reasoning for RDF Validation. In *Proceedings of the 11th International Conference on Semantic Systems (SEMANTiCS 2015)* (pp. 33–40). Vienna, Austria: ACM. http://doi.acm.org/10.1145/2814864.2814867
2. **Bosch, Thomas**, Nolle, A., Acar, E., & Eckert, K. (2015b). RDF Validation Requirements - Evaluation and Logical Underpinning. *Computing Research Repository (CoRR)*, abs/1501.03933. http://arxiv.org/abs/1501.03933

Chapter 9:
Evaluating the Usability of Constraint Types for Assessing RDF Data Quality

1. **Hartmann, Thomas**, Zapilko, B., Wackerow, J., & Eckert, K. (2015a). Constraints to Validate RDF Data Quality on Common Vocabularies in the Social, Behavioral, and Economic Sciences. *Computing Research Repository (CoRR)*, abs/1504.04479. http://arxiv.org/abs/1504.04479
2. **Hartmann, Thomas**, Zapilko, B., Wackerow, J., & Eckert, K. (2015b). Evaluating the Quality of RDF Data Sets on Common Vocabularies in the Social, Behavioral, and Economic Sciences. *Computing Research Repository (CoRR)*, abs/1504.04478. http://arxiv.org/abs/1504.04478
3. **Hartmann, Thomas**, Zapilko, B., Wackerow, J., & Eckert, K. (2016). Validating RDF Data Quality using Constraints to Direct the Development of Constraint Languages. In *Proceedings of the 10th International Conference on Semantic Computing (ICSC 2016)* Laguna Hills, California, USA: IEEE. http://www.ieee-icsc.com/

G.2 Publications by Publication Type

Journal Articles

1. **Bosch, Thomas** & Mathiak, B. (2015). Use Cases Related to an Ontology of the Data Documentation Initiative. *IASSIST Quarterly*, 38(4) & 39(1), 25–37. http://iassistdata.org/iq/issue/38/4
2. **Bosch, Thomas**, Olsson, O., Gregory, A., & Wackerow, J. (2015c). DDI-RDF Discovery - A Discovery Model for Microdata. *IASSIST Quarterly*, 38(4) & 39(1), 17–24. http://iassistdata.org/iq/issue/38/4
3. **Bosch, Thomas** & Zapilko, B. (2015). Semantic Web Applications for the Social Sciences. *IASSIST Quarterly*, 38(4) & 39(1), 7–16. http://iassistdata.org/iq/issue/38/4
4. Schaible, J., Zapilko, B., **Bosch, Thomas**, & Zenk-Möltgen, W. (2015). Linking Study Descriptions to the Linked Open Data Cloud. *IASSIST Quarterly*, 38(4) & 39(1), 38–46. http://iassistdata.org/iq/issue/38/4
5. **Bosch, Thomas** & Mathiak, B. (2013b). How to Accelerate the Process of Designing Domain Ontologies based on XML Schemas. *International Journal of Metadata, Semantics and Ontologies - Special Issue on Metadata, Semantics and Ontologies for Web Intelligence*, 8(3), 254 – 266. http://www.inderscience.com/info/inarticle.php?artid=57760

Articles in Conference Proceedings

1. **Hartmann, Thomas**, Zapilko, B., Wackerow, J., & Eckert, K. (2016). Validating RDF Data Quality using Constraints to Direct the Development of Constraint Languages. In *Proceedings of the 10th International*

Conference on Semantic Computing (ICSC 2016) Laguna Hills, California, USA: IEEE. http://www.ieee-icsc.com/

2. **Bosch, Thomas** & Eckert, K. (2015). Guidance, Please! Towards a Framework for RDF-based Constraint Languages. In *Proceedings of the 15th DCMI International Conference on Dublin Core and Metadata Applications (DC 2015)* São Paulo, Brazil. http://dcevents.dublincore.org/IntConf/dc-2015/paper/view/386/368
3. **Bosch, Thomas**, Acar, E., Nolle, A., & Eckert, K. (2015a). The Role of Reasoning for RDF Validation. In *Proceedings of the 11th International Conference on Semantic Systems (SEMANTiCS 2015)* (pp. 33–40). Vienna, Austria: ACM. http://doi.acm.org/10.1145/2814864.2814867
4. **Bosch, Thomas** & Eckert, K. (2014a). Requirements on RDF Constraint Formulation and Validation. In *Proceedings of the 14th DCMI International Conference on Dublin Core and Metadata Applications (DC 2014)* Austin, Texas, USA. http://dcevents.dublincore.org/IntConf/dc-2014/paper/view/257
5. **Bosch, Thomas** & Eckert, K. (2014b). Towards Description Set Profiles for RDF using SPARQL as Intermediate Language. In *Proceedings of the 14th DCMI International Conference on Dublin Core and Metadata Applications (DC 2014)* Austin, Texas, USA. http://dcevents.dublincore.org/IntConf/dc-2014/paper/view/270
6. **Bosch, Thomas**, Cyganiak, R., Wackerow, J., & Zapilko, B. (2012). Leveraging the DDI Model for Linked Statistical Data in the Social, Behavioural, and Economic Sciences. In *Proceedings of the 12th DCMI International Conference on Dublin Core and Metadata Applications (DC 2012)* Kuching, Sarawak, Malaysia. http://dcpapers.dublincore.org/pubs/article/view/3654
7. **Bosch, Thomas** (2012). Reusing XML Schemas' Information as a Foundation for Designing Domain Ontologies. In P. Cudré-Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J. Parreira, J. Hendler, G. Schreiber, A. Bernstein, & E. Blomqvist (Eds.), *The Semantic Web - ISWC 2012*, volume 7650 of *Lecture Notes in Computer Science* (pp. 437–440). Springer Berlin Heidelberg. http://dx.doi.org/10.1007/978-3-642-35173-0_34
8. **Bosch, Thomas** & Mathiak, B. (2012). XSLT Transformation Generating OWL Ontologies Automatically Based on XML Schemas. In *Proceedings of the 6th International Conference for Internet Technology and Secured Transactions (ICITST 2011), IEEE Xplore Digital Library* (pp. 660–667). Abu Dhabi, United Arab Emirates. http://edas.info/web/icitst2011/program.html
9. **Bosch, Thomas**, Wira-Alam, A., & Mathiak, B. (2011). Designing an Ontology for the Data Documentation Initiative. In *Proceedings of the 8th Extended Semantic Web Conference (ESWC 2011), Poster-Session* Heraklion, Greece. http://www.eswc2011.org/content/accepted-posters.html

Articles in Workshop Proceedings

1. **Bosch, Thomas**, Cyganiak, R., Gregory, A., & Wackerow, J. (2013a). DDI-RDF Discovery Vocabulary: A Metadata Vocabulary for Documenting Research and Survey Data. In *Proceedings of the 6th Workshop on Linked Data on the Web (LDOW 2013), 22nd International World Wide Web Conference (WWW 2013)*, volume 996 Rio de Janeiro, Brazil. http://ceur-ws.org/Vol-996/
2. **Bosch, Thomas**, Zapilko, B., Wackerow, J., & Gregory, A. (2013b). Towards the Discovery of Person-Level Data - Reuse of Vocabularies and Related Use Cases. In *Proceedings of the 1st International Workshop on Semantic Statistics (SemStats 2013), 12th International Semantic Web Conference (ISWC 2013),* Sydney, Australia. http://semstats.github.io/2013/proceedings
3. **Bosch, Thomas** & Mathiak, B. (2011). Generic Multilevel Approach Designing Domain Ontologies Based on XML Schemas. In *Proceedings of the 1st Workshop Ontologies Come of Age in the Semantic Web (OCAS 2011), 10th International Semantic Web Conference (ISWC 2011)* (pp. 1–12). Bonn, Germany. http://ceur-ws.org/Vol-809/

Specifications

1. **Bosch, Thomas**, Cyganiak, R., Wackerow, J., & Zapilko, B. (2016). *DDI-RDF Discovery Vocabulary: A Vocabulary for Publishing Metadata about Data Sets (Research and Survey Data) into the Web of Linked Data*. DDI Alliance Specification, DDI Alliance. http://rdf-vocabulary.ddialliance.org/discovery
2. Wackerow, J., Hoyle, L., & **Bosch, Thomas** (2016). *Physical Data Description*. DDI Alliance Specification, DDI Alliance. http://rdf-vocabulary.ddialliance.org/phdd.html

Technical Reports

1. Vompras, J., Gregory, A., **Bosch, Thomas**, & Wackerow, J. (2015). Scenarios for the DDI-RDF Discovery Vocabulary. *DDI Working Paper Series*. http://dx.doi.org/10.3886/DDISemanticWeb02
2. Alonen, M., **Bosch, Thomas**, Charles, V., Clayphan, R., Coyle, K., Dröge, E., Isaac, A., Matienzo, M., Pohl, A., Rühle, S., & Svensson, L. (2015b). *Report on Validation Requirements*. DCMI Draft, Dublin Core Metadata Initiative (DCMI). http://wiki.dublincore.org/index.php/RDF_Application_Profiles/Requirements
3. Alonen, M., **Bosch, Thomas**, Charles, V., Clayphan, R., Coyle, K., Dröge, E., Isaac, A., Matienzo, M., Pohl, A., Rühle, S., & Svensson, L. (2015a). *Report on the Current State: Use Cases and Validation Requirements*. DCMI Draft, Dublin Core Metadata Initiative (DCMI). http://wiki.dublincore.org/index.php/RDF_Application_Profiles/UCR_Deliverable

4. **Bosch, Thomas**, Nolle, A., Acar, E., & Eckert, K. (2015b). RDF Validation Requirements - Evaluation and Logical Underpinning. *Computing Research Repository (CoRR)*, abs/1501.03933. http://arxiv.org/abs/1501.03933
5. **Hartmann, Thomas**, Zapilko, B., Wackerow, J., & Eckert, K. (2015a). Constraints to Validate RDF Data Quality on Common Vocabularies in the Social, Behavioral, and Economic Sciences. *Computing Research Repository (CoRR)*, abs/1504.04479. http://arxiv.org/abs/1504.04479
6. **Hartmann, Thomas**, Zapilko, B., Wackerow, J., & Eckert, K. (2015b). Evaluating the Quality of RDF Data Sets on Common Vocabularies in the Social, Behavioral, and Economic Sciences. *Computing Research Repository (CoRR)*, abs/1504.04478. http://arxiv.org/abs/1504.04478
7. **Bosch, Thomas**, Wira-Alam, A., & Mathiak, B. (2014). Designing an Ontology for the Data Documentation Initiative. *Computing Research Repository (CoRR)*, abs/1402.3470. http://arxiv.org/abs/1402.3470
8. **Bosch, Thomas** & Mathiak, B. (2013a). *Evaluation of a Generic Approach for Designing Domain Ontologies Based on XML Schemas*. Gesis Technical Report 08, Gesis - Leibniz Institute for the Social Sciences, Mannheim, Germany. http://www.gesis.org/publikationen/archiv/gesis-technical-reports/
9. Block, W., **Bosch, Thomas**, Fitzpatrick, B., Gillman, D., Greenfield, J., Gregory, A., Hebing, M., Hoyle, L., Humphrey, C., Johnson, J., Linnerud, J., Mathiak, B., McEachern, S., Radler, B., Risnes, Ø., Smith, D., Thomas, W., Wackerow, J., Wegener, D., & Zenk-Möltgen, W. (2012). Developing a Model-Driven DDI Specification. *DDI Working Paper Series*

References

[1] Alonen, M., **Bosch, Thomas**, Charles, V., Clayphan, R., Coyle, K., Dröge, E., Isaac, A., Matienzo, M., Pohl, A., Rühle, S., & Svensson, L. (2015a). *Report on the Current State: Use Cases and Validation Requirements*. DCMI Draft, Dublin Core Metadata Initiative (DCMI). http://wiki.dublincore.org/index.php/RDF_Application_Profiles/UCR_Deliverable.

[2] Alonen, M., **Bosch, Thomas**, Charles, V., Clayphan, R., Coyle, K., Dröge, E., Isaac, A., Matienzo, M., Pohl, A., Rühle, S., & Svensson, L. (2015b). *Report on Validation Requirements*. DCMI Draft, Dublin Core Metadata Initiative (DCMI). http://wiki.dublincore.org/index.php/RDF_Application_Profiles/Requirements.

[3] Apache Software Foundation (2015). *Apache Log4j 2 v. 2.3 User's Guide*. Technical report, Apache Software Foundation. http://logging.apache.org/log4j/2.x/log4j-users-guide.pdf.

[4] Block, W., **Bosch, Thomas**, Fitzpatrick, B., Gillman, D., Greenfield, J., Gregory, A., Hebing, M., Hoyle, L., Humphrey, C., Johnson, J., Linnerud, J., Mathiak, B., McEachern, S., Radler, B., Risnes, Ø., Smith, D., Thomas, W., Wackerow, J., Wegener, D., & Zenk-Möltgen, W. (2012). Developing a Model-Driven DDI Specification. *DDI Working Paper Series*.

[5] **Bosch, Thomas** (2012). Reusing XML Schemas' Information as a Foundation for Designing Domain Ontologies. In P. Cudré-Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J. Parreira, J. Hendler, G. Schreiber, A. Bernstein, & E. Blomqvist (Eds.), *The Semantic Web - ISWC 2012*, volume 7650 of *Lecture Notes in Computer Science* (pp. 437–440). Springer Berlin Heidelberg. http://dx.doi.org/10.1007/978-3-642-35173-0_34.

[6] **Bosch, Thomas**, Acar, E., Nolle, A., & Eckert, K. (2015a). The Role of Reasoning for RDF Validation. In *Proceedings of the 11th International Conference on Semantic Systems (SEMANTiCS 2015)* (pp. 33–40). Vienna, Austria: ACM. http://doi.acm.org/10.1145/2814864.2814867.

[7] **Bosch, Thomas**, Cyganiak, R., Gregory, A., & Wackerow, J. (2013a). DDI-RDF Discovery Vocabulary: A Metadata Vocabulary for Document-

ing Research and Survey Data. In *Proceedings of the 6th Workshop on Linked Data on the Web (LDOW 2013), 22nd International World Wide Web Conference (WWW 2013)*, volume 996 Rio de Janeiro, Brazil. http://ceur-ws.org/Vol-996/.

[8] **Bosch, Thomas**, Cyganiak, R., Wackerow, J., & Zapilko, B. (2012). Leveraging the DDI Model for Linked Statistical Data in the Social, Behavioural, and Economic Sciences. In *Proceedings of the 12th DCMI International Conference on Dublin Core and Metadata Applications (DC 2012)* Kuching, Sarawak, Malaysia. http://dcpapers.dublincore.org/pubs/article/view/3654.

[9] **Bosch, Thomas**, Cyganiak, R., Wackerow, J., & Zapilko, B. (2016). *DDI-RDF Discovery Vocabulary: A Vocabulary for Publishing Metadata about Data Sets (Research and Survey Data) into the Web of Linked Data.* DDI Alliance Specification, DDI Alliance. http://rdf-vocabulary.ddialliance.org/discovery.

[10] **Bosch, Thomas** & Eckert, K. (2014a). Requirements on RDF Constraint Formulation and Validation. In *Proceedings of the 14th DCMI International Conference on Dublin Core and Metadata Applications (DC 2014)* Austin, Texas, USA. http://dcevents.dublincore.org/IntConf/dc-2014/paper/view/257.

[11] **Bosch, Thomas** & Eckert, K. (2014b). Towards Description Set Profiles for RDF using SPARQL as Intermediate Language. In *Proceedings of the 14th DCMI International Conference on Dublin Core and Metadata Applications (DC 2014)* Austin, Texas, USA. http://dcevents.dublincore.org/IntConf/dc-2014/paper/view/270.

[12] **Bosch, Thomas** & Eckert, K. (2015). Guidance, Please! Towards a Framework for RDF-based Constraint Languages. In *Proceedings of the 15th DCMI International Conference on Dublin Core and Metadata Applications (DC 2015)* São Paulo, Brazil. http://dcevents.dublincore.org/IntConf/dc-2015/paper/view/386/368.

[13] **Bosch, Thomas** & Mathiak, B. (2011). Generic Multilevel Approach Designing Domain Ontologies Based on XML Schemas. In *Proceedings of the 1st Workshop Ontologies Come of Age in the Semantic Web (OCAS 2011), 10th International Semantic Web Conference (ISWC 2011)* (pp. 1–12). Bonn, Germany. http://ceur-ws.org/Vol-809/.

[14] **Bosch, Thomas** & Mathiak, B. (2012). XSLT Transformation Generating OWL Ontologies Automatically Based on XML Schemas. In *Proceedings of the 6th International Conference for Internet Technology and Secured Transactions (ICITST 2011), IEEE Xplore Digital Library* (pp. 660–667). Abu Dhabi, United Arab Emirates. http://edas.info/web/icitst2011/program.html.

[15] **Bosch, Thomas** & Mathiak, B. (2013a). *Evaluation of a Generic Approach for Designing Domain Ontologies Based on XML Schemas.* Gesis Technical Report 08, Gesis - Leibniz Institute for the Social Sciences,

Mannheim, Germany. http://www.gesis.org/publikationen/archiv/gesis-technical-reports/.

[16] **Bosch, Thomas** & Mathiak, B. (2013b). How to Accelerate the Process of Designing Domain Ontologies based on XML Schemas. *International Journal of Metadata, Semantics and Ontologies - Special Issue on Metadata, Semantics and Ontologies for Web Intelligence*, 8(3), 254 – 266. http://www.inderscience.com/info/inarticle.php?artid=57760.

[17] **Bosch, Thomas** & Mathiak, B. (2015). Use Cases Related to an Ontology of the Data Documentation Initiative. *IASSIST Quarterly*, 38(4) & 39(1), 25–37. http://iassistdata.org/iq/issue/38/4.

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

[19] **Bosch, Thomas**, Olsson, O., Gregory, A., & Wackerow, J. (2015c). DDI-RDF Discovery - A Discovery Model for Microdata. *IASSIST Quarterly*, 38(4) & 39(1), 17–24. http://iassistdata.org/iq/issue/38/4.

[20] **Bosch, Thomas**, Wira-Alam, A., & Mathiak, B. (2011). Designing an Ontology for the Data Documentation Initiative. In *Proceedings of the 8th Extended Semantic Web Conference (ESWC 2011), Poster-Session* Heraklion, Greece. http://www.eswc2011.org/content/accepted-posters.html.

[21] **Bosch, Thomas**, Wira-Alam, A., & Mathiak, B. (2014). Designing an Ontology for the Data Documentation Initiative. *Computing Research Repository (CoRR)*, abs/1402.3470. http://arxiv.org/abs/1402.3470.

[22] **Bosch, Thomas** & Zapilko, B. (2015). Semantic Web Applications for the Social Sciences. *IASSIST Quarterly*, 38(4) & 39(1), 7–16. http://iassistdata.org/iq/issue/38/4.

[23] **Bosch, Thomas**, Zapilko, B., Wackerow, J., & Gregory, A. (2013b). Towards the Discovery of Person-Level Data - Reuse of Vocabularies and Related Use Cases. In *Proceedings of the 1st International Workshop on Semantic Statistics (SemStats 2013), 12th International Semantic Web Conference (ISWC 2013),* Sydney, Australia. http://semstats.github.io/2013/proceedings.

[24] **Hartmann, Thomas**, Zapilko, B., Wackerow, J., & Eckert, K. (2015a). Constraints to Validate RDF Data Quality on Common Vocabularies in the Social, Behavioral, and Economic Sciences. *Computing Research Repository (CoRR)*, abs/1504.04479. http://arxiv.org/abs/1504.04479.

[25] **Hartmann, Thomas**, Zapilko, B., Wackerow, J., & Eckert, K. (2015b). Evaluating the Quality of RDF Data Sets on Common Vocabularies in the Social, Behavioral, and Economic Sciences. *Computing Research Repository (CoRR)*, abs/1504.04478. http://arxiv.org/abs/1504.04478.

[26] **Hartmann, Thomas**, Zapilko, B., Wackerow, J., & Eckert, K. (2016). Validating RDF Data Quality using Constraints to Direct the Development of Constraint Languages. In *Proceedings of the 10th International Confer-*

ence on Semantic Computing (ICSC 2016) Laguna Hills, California, USA: IEEE. http://www.ieee-icsc.com/.

[27] Krötzsch, M., Simančík, F., & Horrocks, I. (2012). A Description Logic Primer. In J. Lehmann & J. Völker (Eds.), *Perspectives on Ontology Learning*. IOS Press.

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

[29] Mader, C., Haslhofer, B., & Isaac, A. (2012). Finding Quality Issues in SKOS Vocabularies. In *Proceedings of the Second International Conference on Theory and Practice of Digital Libraries*, TPDL'12 (pp. 222–233). Berlin, Heidelberg: Springer-Verlag. http://link.springer.com/chapter/10.1007%2F978-3-642-33290-6_25.

[30] Reutter, J. L., Soto, A., & Vrgoč, D. (2015). Recursion in SPARQL. In M. Arenas, O. Corcho, E. Simperl, M. Strohmaier, M. d'Aquin, K. Srinivas, P. Groth, M. Dumontier, J. Heflin, K. Thirunarayan, K. Thirunarayan, & S. Staab (Eds.), *The Semantic Web - ISWC 2015*, volume 9366 of *Lecture Notes in Computer Science* (pp. 19–35). Springer International Publishing.

[31] Schaible, J., Zapilko, B., **Bosch, Thomas**, & Zenk-Möltgen, W. (2015). Linking Study Descriptions to the Linked Open Data Cloud. *IASSIST Quarterly*, 38(4) & 39(1), 38–46. http://iassistdata.org/iq/issue/38/4.

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

[33] Vompras, J., Gregory, A., **Bosch, Thomas**, & Wackerow, J. (2015). Scenarios for the DDI-RDF Discovery Vocabulary. *DDI Working Paper Series*. http://dx.doi.org/10.3886/DDISemanticWeb02.

[34] Wackerow, J., Hoyle, L., & **Bosch, Thomas** (2016). *Physical Data Description*. DDI Alliance Specification, DDI Alliance. http://rdf-vocabulary.ddialliance.org/phdd.html.