Validation Framework for RDF-based Constraint Languages

Zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften

Dr.-Ing.

von der Fakultät für Wirtschaftswissenschaften
des Karlsruher Instituts für Technologie (KIT)

genehmigte

DISSERTATION

von

M.Sc. (TUM) Thomas Hartmann

Tag der mündlichen Prüfung:	08.07.2016
Referent:	Prof. Dr. York Sure-Vetter
Korreferent:	Prof. Dr. Kai Eckert Hochschule der Medien, Stuttgart

Karlsruhe Juli 2016

To My Love Corinna

Abstract

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. Three missing vocabularies have been developed 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. We propose a general approach 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, without having any 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 derived from case studies and use cases provided by various data institutions. We use these constraint types to gain a better understanding of the expressiveness of solutions, investigate the role that reasoning plays in practical data validation, and give directions for the further development of constraint languages.

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 with 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.

We evaluate the usability of constraint types for assessing RDF data quality by collecting and classifying constraints on common vocabularies and 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 future development of constraint languages.

Acknowledgements

A PhD thesis, although single authored, is a collaborative effort. I would like to thank each person who supported me creating this thesis. First and foremost, I want to express my profound gratitude to my advisors Prof. Dr. York Sure-Vetter and Prof. Dr. Kai Eckert. *Prof. Dr. York Sure-Vetter*, former president of GESIS, the Leibniz Institute for the Social Sciences, and Prof. Dr. Christof Wolf, current president of GESIS, are the two persons having primarily been responsible for my decision to work as research associate for the research institute GESIS. Prof. Dr. York Sure-Vetter always motivated me to submit research papers to top conferences in the field of Semantic Web and Linked Data. I offer my sincere thanks to him for taking the time to thoroughly review my thesis, giving me invaluable feedback especially during various PhD symposiums and in the last highly productive months, and of course for offering me the possibility to graduate at the Karlsruhe Institute of Technology.

I am very proud of having *Prof. Dr. Kai Eckert* as my co-advisor as he has laid the foundation on which this thesis has been built. During a flight to Austin, where we held a presentation at an international conference in 2014, we had very inspiring discussions which, in the end, have led to the basic concept of this thesis. By providing the perspective on the cultural heritage sector, he has given me extraordinary stimulating feedback throughout my research work, contributed to many innovative ideas, and significantly improved my scientific writing skills by co-writing many research papers. He always motivated me to work in several international working groups to collaboratively share thoughts on topics closely related to my individual research.

I would like to give my deepest gratitude to my mentor and team leader at GESIS - *Joachim Wackerow*. Having the perspective on the social, behavioral, and economic sciences, he constantly made me contribute to various workshops, was always present with his sound and experienced advice, and continuously offered me encouragement. My particular thanks go to him for countless stimulating whiteboard sessions, constructive discussions, valuable

feedback within various contexts, and for taking the time to review parts of my thesis.

I am particularly grateful to *Prof. Dr. Christof Wolf* who has given me the chance to take the technical responsibility for four IT projects when I have been research associate at GESIS. From the beginning, he believed in my willingness and technical skills that enabled me to decisively contribute to the success of these projects. Being president of GESIS and head of the department Monitoring Society and Social Change, he made it possible for me to attend 19 international conferences and 23 international workshops in 16 countries.

For their especially important contributions to my research, I thank all my co-authors of the publications that form the basis of this thesis - particular thanks to *Prof. Dr. Kai Eckert*, *Joachim Wackerow*, and *Dr. Benjamin Zapilko*. Furthermore, I would like to thank *Erman Acar* from the University of Mannheim for numerous paper sessions during Christmas time in 2014.

Playing an important role in four international working groups has been a perfect training ground for my personal communication, networking, and team skills. That's why I want to offer my thanks to all members of those working groups for their great collaborative work; especially to the chairs *Karen Coyle*, *Antoine Isaak*, *Joachim Wackerow*, *Arnaud Le Hors*, *Franck Cotton*, and *Adam Brown*.

Special thanks go to *Prof. Dr. Brigitte Mathiak* from the University of Cologne and *Oliver Hopt* who have worked with me on the initial idea for the RDFication of XML, which has built the ground for further research in this part of the thesis.

It was a great pleasure working with the team members of the MISSY projects *Prof. Dr. Christof Wolf*, *Dr. Jeanette Bohr*, *Andias Wira-Alam*, *Oliver Hopt*, *Karin Schuller*, *Florian Thirolf*, *Dr. Claus-Peter Klas*, *Alina Weber*, and in particular *Matthäus Zloch*, an extraordinary helpful colleague and friend from whom I was able to learn so much about Web application development and agile project management.

For providing a very fruitful working atmosphere, my heartfelt thanks go to all *colleagues at GESIS*, especially from the *German Microdata Lab* team and the department *Monitoring Society and Social Change* in Mannheim as well as the department *Knowledge Technologies for the Social Sciences* in Cologne. My friends *Johann 'Wanja' Schaible*, *Matthäus Zloch*, *Dimitar Dimitrov*, *Andias Wira-Alam*, *Dr. Benjamin Zapilko*, and *Alexander Mühlbauer* are some of them who influenced and enhanced my work.

My parents *Ruth* and *Franz*, my brother *Daniel*, and my parents-in-law *Margit* and *Peter* receive my deepest gratitude and love for the many years of moral support and especially for always believing in me. I warmly thank all my *close friends* who are still there, even though I have made myself very scarce in the last six years. And last but not least, I cordially thank my beloved wife *Corinna* for her love and understanding that strongly encouraged me and, in the end, made this thesis possible. I am so lucky and grateful to have you in my life! I will always love you!

Contents

List of Figures

List of Tables

1

Introduction

The Semantic Web [22, 283] is an open knowledge space where knowledge can be represented and new information can be gained by performing reasoning over heterogeneous data sources. Linked Data interweaves the Semantic Web with the World Wide Web of documents [18–21], focuses on the strength of the Semantic Web to exchange and integrate data from various sources [145], and thus forms the Giant Global Graph. The notion of Linked (Open) Data and its principles [144] clearly increased the acceptance – not to say the excitement – of data providers for the underlying Semantic Web technologies. Breaking down the data to statements and globally unique identifiers for subjects, predicates, and objects facilitates the merging of arbitrary data sets without conflicts, while the exploitation of links between the identifiers brings together what belongs together. Early concerns of the data providers regarding stability and trustability of the data have been addressed and largely been solved, not only by technical means regarding versioning and provenance, but also by the providers getting accustomed to the open data world with its peculiarities.

Linked Data, however, is still not the primary means to create, store, and manage data on the side of the providers. Linked Data is mostly offered as only one view on the data, a one-way road, disconnected from the internal data representation. To the obstacles for the full adoption of RDF, possibly comparable to XML, belongs the lack of accepted ways to formulate (local) constraints on RDF data and validate the data according to these constraints.

The Common Need of Data Practitioners for RDF Validation

For many data practitioners, data libraries, data archives, and research institutes embracing the world of Linked Data, the openness and flexibility is a mixed blessing. For them, 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. Among the reasons for the success of XML is the possibility to formulate fine-grained constraints to be met by the data and to validate data using powerful systems like Document Type Definition, XML Schema, RELAX NG, and Schematron.

A typical example is the library domain that co-developed and adopted Linked Data principles very early. Libraries have a long tradition in developing and using interoperable data formats. For libraries, the common description of resources is key business, so the definition of library records describing books. There are clear rules which information has to be available to describe a book properly (e.g., ISBN, title, and author), but also how information like an ISBN number is correctly represented. Libraries seek to make their own data reusable for general purposes, but also to enrich and interlink their data. Checking if third-party data meets own requirements or validating existing data according to new needs for a Linked Data application are among common case studies for RDF validation. While they appreciate the openness of Linked Data and the data modeling principles provided by RDF, their data is still mostly represented in XML and this is unlikely to change soon.

Data practitioners of various domains like the cultural heritage sector and the social, behavioral, and economic sciences are used to represent their data in XML. To publish, interlink, and integrate their data, they simply map it to RDF. It is unlikely, at least in the short to medium term, that they will completely move to RDF and thereby change their behavior they have been successful with over decades. Representing data in XML and RDF, however, allows data practitioners (1) to stay in the generally accepted and proven XML environment they are familiar with, (2) to benefit from emerging and promising Semantic Web and Linked Data technologies, and (3) to improve the quality of their data by using common RDF validation tools.

RDF Validation as Research Field

Recently, RDF validation as a research field gained speed due to common needs of data practitioners. In 2013, the W3C organized the *RDF Validation Workshop*[1] [202], where experts from industry, government, and academia presented and discussed first case studies for constraint formulation and RDF data validation. Two working groups on RDF validation that follow up on this workshop have been established in 2014 to develop a language for expressing constraints on RDF data and defining structural constraints on RDF graphs: the *W3C RDF Data Shapes Working Group*[2] (40 participants from 24 organizations) and the *Dublin Core Metadata Initiative (DCMI) RDF Application Profiles Task Group*[3] (31 persons from 23 organizations).

The DCMI working group bundles the requirements of data institutions of the cultural heritage sector (mostly from the library domain) and the social,

[1] http://www.w3.org/2012/12/rdf-val
[2] https://www.w3.org/2014/data-shapes/charter
[3] http://wiki.dublincore.org/index.php/RDF-Application-Profiles

behavioral, and economic sciences and represents them in the W3C working group. The intention of the DCMI working group is to (1) gather case studies from data professionals having RDF validation related problems in their individual contexts, (2) extract functional use cases out of these case studies, (3) specify requirements to be fulfilled to adequately solve these problems and meet the use cases, (4) investigate existing best-practices regarding the coverage of these requirements, and (5) identify gaps and recommend best-practices to close them, i.e., practical solutions for the representation of application profiles, including the formulation and checking of data constraints.

In a heterogeneous environment like the Web, there is not necessarily a one-size-fits-all solution, especially as existing solutions should rather be integrated than replaced, not least to avoid long and fruitless discussions about the "best" approach. For constraint formulation and RDF data validation, several languages exist or are currently developed. The *SPARQL Query Language for RDF* [134], the *SPARQL Inferencing Notation (SPIN)* [185], the *Web Ontology Language (OWL)* [27], *Shape Expressions (ShEx)* [287], *Resource Shapes (ReSh)* [274], and *Description Set Profiles (DSP)* [234] are the six most promising and widely used constraint languages that are most popular among data practitioners. With its direct support of validation via SPARQL, SPIN is very popular and certainly plays an important role for future developments in this field. Despite the fact that OWL is arguably not a constraint language, it is widely used in practice as such under the closed-world and unique name assumptions. In addition to these already existing constraint languages, the W3C working group currently develops the *Shapes Constraint Language (SHACL)* [186], an RDF vocabulary for describing RDF graph structures. Yet, there is no clear favorite and none of these languages is able to meet all requirements raised by data practitioners. This is the reason why further research on RDF validation and the development of constraint languages is needed.

1.1 Objectives and Structure of the Thesis

Figure 1.1 shows the structure of the thesis, how the chapters are interrelated, their main contributions, and how they are related to individual research questions.

Vocabularies for Representing Research Data and its Metadata

The *social, behavioral, and economic (SBE) sciences* require high-quality data for their empirical research. For more than a decade, members of the SBE sciences community have been developing and using a metadata standard, composed of almost twelve hundred metadata fields, known as the *Data Documentation Initiative (DDI)* [93, 94], an XML format to disseminate, manage,

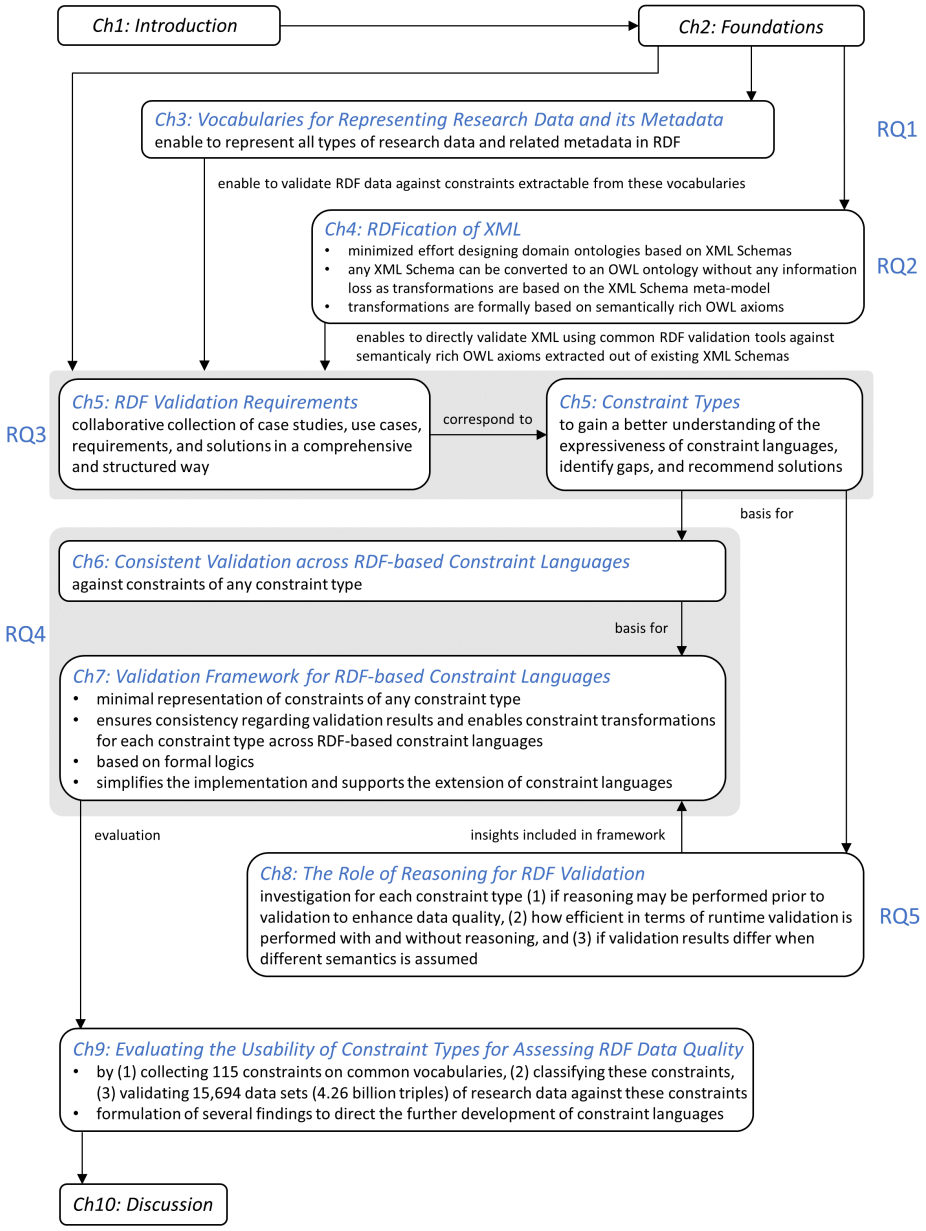

Fig. 1.1. Structure of the Thesis

and reuse data collected and archived for research [316]. In XML, the definition of schemas containing constraints on data and the validation of data according to these constraints is commonly used to ensure a certain level of data quality. With the rise of the Web of Data, data professionals and institutions are very interested in having their data be discovered and used by publishing their data directly in RDF or at least accurate metadata about their data to facilitate data integration.

There are different types of research data and related metadata. Because of the lack of respective RDF vocabularies, however, just a few of them can be expressed in RDF. We have developed three missing vocabularies (1) to represent all types of research data and its metadata in RDF and (2) to validate RDF data against constraints extractable from these vocabularies (see Chapter 3):

- The *DDI-RDF Discovery Vocabulary (DDI-RDF)* to support the discovery of metadata on *unit-record data*, i.e., data collected about individuals, businesses, and households.
- *Physical Data Description (PHDD)*, a vocabulary to describe data in tabular format and its physical properties.
- The *SKOS Extension for Statistics (XKOS)*, a vocabulary to describe formal statistical classifications and introduce refinements of SKOS semantic properties to allow the use of more specific relations between concepts.

RDFication of XML Enabling to use RDF Validation Technologies

There is a huge amount of XML Schemas describing conceptual models about data in various domains, but compared to XML Schemas there are still only a few ontologies formally representing the intended semantics of particular domains. This is the reason why the LOD cloud is still missing conceptual descriptions [169]. In the B2B domain, e.g., there are hundreds of XML Schemas to encode exchanged XML data, but not many ontologies.

Data practitioners 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 data against semantically rich OWL axioms when using them in terms of constraints and extracting them from XML Schemas adequately representing certain domains, we propose on formal logics and the XML Schema meta-model based automatic transformations of arbitrary XML Schemas and conforming XML documents into OWL ontologies and corresponding RDF data without any information loss. This does not cause any additional manual effort, as these constraints have already been defined by the XML community within underlying XML Schemas.

As generated ontologies are not conform to the highest quality requirements of more sophisticated domain ontologies regarding the intended semantics of given domains, we automatically derive domain ontologies out of

generated ontologies using manually defined SWRL rules. This way, we (1) reduce the complexity of generated ontologies and therefore underlying XML Schemas and (2) further supplement OWL axioms with additional domain-specific semantic information not or not satisfyingly covered by underlying XML Schemas.

By evaluating the proposed approach, we verify the hypothesis that the effort and the time needed to deliver high quality domain ontologies from scratch by reusing information of already existing XML Schemas properly delineating particular domains is much less than creating domain ontologies completely manually and anew. The resulting domain ontologies are as usable as ontologies that are completely constructed by hand, but with a fraction of necessary effort (see Chapter 4).

RDF Validation Requirements and Constraint Types

Our work is supposed to lay the ground for subsequent activities in the W3C and DCMI working groups. We propose to relate existing solutions to case studies and use cases by means of requirements, extracted from the latter and fulfilled by the former. We therefore collected the findings of the RDF Validation Workshop and the working groups and initiated a community-driven database of requirements to formulate constraints and validate RDF data. Additionally, we added requirements from other sources, particularly in the form of constraint types that are supported by existing approaches, e.g., those expressible in OWL 2. 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.

Laying the ground on cooperatively collected case studies and relating solutions to case studies and use cases by means of requirements makes sure that (1) commonly approved requirements cover real world needs of data professionals having RDF validation related problems and (2) the further development of constraint languages is based on universally accepted requirements.

Based on our work in these working groups and the jointly identified requirements, we have published by today 81 types of constraints that are required by various 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 a specific requirement in the database. We use this collection of constraint types to gain a better understanding of the expressiveness of existing and currently developed 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 (see Chapter 5).

Consistent Validation across RDF-based Constraint Languages

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

We use SPIN, a SPARQL-based way to formulate and check constraints, as basic validation framework and present a general approach how RDF-based constraint languages can be executed on RDF data in a consistent way, i.e., how to consistently implement the validation of RDF data against constraints of any constraint type in any RDF-based language using SPARQL as an intermediate language. This is necessary since (1) multiple implementations using distinct underlying technologies hamper the interoperability of constraint languages and (2) full and differing implementations of several languages are hard to maintain for solution providers. We claim and provide evidence from literature that constraints of each type in any RDF-based language can be checked with plain SPARQL as low-level execution language.

We have developed a validation environment which is online available at http://purl.org/net/rdfval-demo and which can be used to validate RDF data on constraints of any type expressed in arbitrary RDF-based constraint languages. To demonstrate the general applicability of the approach, we provide implementations for (1) all OWL 2 and DSP language constructs, (2) all constraint types which are expressible in OWL 2 and DSP, and (3) major constraint types representable by ReSh and ShEx (see Chapter 6).

Validation Framework for RDF-based Constraint Languages

Several solutions to formulate constraints and validate RDF data against constraints exist or are currently developed. However, none of the languages, we consider being high-level constraint languages, is able to meet all requirements raised by data practitioners, i.e., enables to express constraints of all constraint types.

We generalize from our experience completely implementing OWL 2 and DSP as well as major constructs of other high-level constraint languages and gained from the analysis of the identified constraint types and introduce a general framework, an abstraction layer that enables to formulate constraints of any type in a way that mappings from high-level constraint languages to the intermediate generic representation can be created more or less straightforwardly. The proposed framework reduces the representation of constraints to the absolute minimum, is based on formal logics, and consists of a very simple conceptual model with a small lightweight vocabulary and constraining elements.

We demonstrate that using another layer on top of SPARQL simplifies the implementation of constraint languages and ensures for any constraint type that (1) whenever semantically equivalent constraints of the same type are checked on RDF data they point out the same set of violations regardless of the language used to express them and (2) semantically equivalent constraints of the same type can be transformed from one RDF-based constraint language to another via the intermediate representation (see Chapter 7).

The Role of Reasoning for RDF Validation

The constraint types form the basis to investigate the role that reasoning and different semantics play 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 performed prior to validation 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. Using these findings, we determine for the most common constraint languages which reasoning constraint types they enable to express and give directions for the further development of constraint languages (see Chapter 8).

Evaluating the Usability of Constraint Types for Assessing RDF Data Quality

A concrete constraint is instantiated from one of the 81 constraint types and defined for a specific vocabulary. We gain a better understanding about the role of certain constraint types for determining the quality of RDF data and therefore evaluate the usability of identified constraint types for assessing RDF data quality by (1) collecting 115 constraints on common vocabularies, either from the vocabularies themselves or from domain experts, (2) classifying these constraints, and (3) validating 15,694 data sets (4.26 billion triples) of research data, obtained from 33 SPARQL endpoints, against these constraints. Based on the large-scale evaluation, we formulate several findings to gain insights and make recommendations to direct the further development of constraint languages (see Chapter 9).

1.2 Research Questions

Research Question 1 *Which types of research data and related metadata are not yet representable in RDF and how to adequately model them to be able to validate RDF data against constraints extractable from these vocabularies?*

The *social, behavioral, and economic (SBE) sciences* require high-quality data for their empirical research. There are different types of research data and related metadata. Because of the lack of respective RDF vocabularies, however, just a few of them can be expressed in RDF. Having these missing vocabularies enables (1) to represent all types of research data and its metadata in RDF and (2) to validate RDF data according to constraints extractable from these vocabularies. The types of research data and related metadata which are not yet expressible in RDF are metadata on (1) unit-record data, (2) tabular data, and (3) formal statistical classifications.

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

Data in tabular format can be represented in form of records with character-separated values (CSV) or with fixed length. CSV files are two-dimensional tables about SBE sciences variables such as sex or age used to capture individual values for persons.

Formal statistical classifications like the International Standard Classification of Occupations are hierarchical concept schemes including concepts, associated numeric codes, short textual labels, definitions, and longer descriptions that include rules for their use. There is a common need to represent the structure and textual properties of classifications as well as different types of relations between classifications and concepts and among concepts.

Research Question 2 *How to directly validate XML data on semantically rich OWL axioms using common RDF validation tools when XML Schemas, adequately representing particular domains, have already been designed?*

Traditionally, domain experts closely work with ontology engineers to design OWL domain ontologies manually from scratch which requires lots of time and effort. In many cases, however, XML Schemas adequately describing certain domains have already been produced by the XML community and all the information contained in XML Schemas can therefore be reused as a basis to design more sophisticated domain ontologies. Saved time and work could be used more effectively to enrich the knowledge representation of given domains with additional domain-specific semantic information not or not satisfyingly covered by already existing XML Schemas.

Data practitioners of many domains still represent their data in XML, but expect to increase the quality of their data by using common RDF validation tools. After automatically transforming XML Schemas and corresponding XML documents into OWL ontologies and conforming RDF data, we are able

to directly validate the data against automatically extracted and semantically rich OWL axioms when using them in terms of constraints. This does not cause any additional manual effort as these constraints have already been defined within underlying XML Schemas. Such constraints could ensure that (1) particular XML elements like the *title* of a book only contain plain text, (2) specific XML attributes such as publication *year* only include positive numbers, or (3) an XML element like *book* only contains listed XML elements (e.g., *isbn*, *title*, and *author*) in a predefined order.

As structures of XML Schemas may be quite complex, it has to be ensured that any XML Schema, i.e., arbitrary complex structures of XML Schemas, can be converted without any information loss. This means that all the information about the terminology of individual domains, the implicit semantics of XML Schema constructs (e.g., class memberships of XML elements and relationships between XML elements), and the syntactic structure of sets of XML documents has to be maintained and modeled with correct semantics in form of suitable OWL axioms.

A main difference between the structural level of XML and the semantic level of RDF is that XML elements are explicitly ordered and RDF is set-oriented. As XML Schemas are commonly used to specify structural relationships of objects in data-centric XML documents, all the structural information like *sequence* and *choice*, determining the syntactic structure of sets of XML documents, has to be preserved. The XML Schema construct *sequence*, e.g., is used to specify the explicit order of XML elements contained in parent XML elements and the XML Schema construct *choice* specifies an exclusive or of its operands.

Research Question 3 *Which types of constraints must be expressible by constraint languages to meet all collaboratively and comprehensively identified requirements to formulate constraints and validate RDF data?*

Our work is supposed to establish a stable foundation for subsequent activities in the working groups on RDF validation. We therefore investigate ways encouraging the collaborative identification of requirements to formulate constraints and validate RDF data against constraints in a comprehensive and structured way within these working groups and beyond.

Just like for the development of standards in general, as in software, we take case studies and use cases as a starting point for the advancement of constraint languages. In case studies, the full background of specific scenarios is described, where the standard or the software is to be applied. Use cases are smaller units where certain actions or typical user inquiries are described. They can be extracted from and thus linked to case studies, but often they are defined directly.

Requirements are extracted out of use cases. They form the basis for development and are used to evaluate existing solutions. Via requirements, solutions get linked to use cases and case studies and it becomes visible which

solutions can be used in a given scenario and what drawbacks might be faced. Our goal is to maintain a set of distinct requirements. Only this way it is possible to evaluate solutions regarding their suitability for use cases and case studies. Use cases can be shared between case studies as well, but this is harder to maintain as use cases are less formal and often more case specific than a requirement. Another goal is a relative stability regarding the requirements, which then can prove to be useful to mediate between data and solution providers.

Laying the ground on case studies collected from various data practitioners and relating solutions to case studies and use cases by means of requirements, extracted from the latter and fulfilled by the former, makes sure that (1) commonly approved requirements cover real world needs of data professionals having RDF validation related problems and (2) the further development of constraint languages is based on universally accepted requirements.

We determine which types of constraints must be expressible by constraint languages to satisfy each of the jointly identified requirements and use this set of constraint types to gain a better understanding of the expressiveness of already 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.

Research Question 4 *How to ensure for any constraint type that (1) RDF data is consistently validated against semantically equivalent constraints of the same constraint type across RDF-based constraint languages and (2) semantically equivalent constraints of the same constraint type can be transformed from one RDF-based constraint language to another?*

None of the solutions, we consider as high-level constraint languages, is able to meet all requirements raised by data practitioners, i.e., enables to express each of the 81 identified constraint types.

High-level constraint languages like ShEx, ReSh, DSP, OWL, and SHACL either lack an implementation to actually validate RDF data according to constraints expressed in these languages or are based on different implementations. This leaves the first subordinate research question how to execute RDF-based constraint languages on RDF data in a consistent way, i.e., how to consistently implement the validation of RDF data against constraints of any constraint type expressed in any RDF-based language. This is necessary since (1) multiple implementations using distinct underlying technologies hamper the interoperability of constraint languages and (2) full and differing implementations of several languages are hard to maintain for solution providers.

4.1 *How to consistently validate RDF data against constraints of any constraint type expressed in any RDF-based constraint language?*

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. Equivalent constraints of that type having exactly the same meaning are expressible in different languages:

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

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

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

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

SHACL: :PublicationShape
          a sh:Shape ;
          sh:scopeClass :Publication ;
          sh:property [
              sh:predicate :author ;
              sh:valueShape :PersonShape ;
              sh:minCount 1 ; ] .
          :PersonShape
              a sh:Shape ;
              sh:scopeClass :Person .
```

When we fully implemented OWL 2 and DSP as well as major constructs of other high-level constraint languages using SPARQL as intermediate language and mappings to SPIN, a SPARQL-based way to specify and check constraints, we found that many mappings actually resemble each other; particularly the mappings of the same constraint type in different languages, but also the mappings of different constraint types, though the latter only on a very superficial, structural level. In the example, it can be seen that the expressions of the high-level constraint languages are comparatively similar - there seems to be a pattern, a common way to express this type of constraints.

We build on the experience gained from mapping several constraint languages to SPIN and the analysis of the identified constraint types to create an intermediate layer, a framework that is able to describe the mechanics of all constraint types in a way that mappings from high-level constraint languages to this intermediate generic representation can be created more or less straight-forwardly.

The basic idea of our framework is very simple: (1) we aim at reducing the representation of constraints to the absolute minimum that has to be provided in a mapping to SPIN and (2) we want to be able to express constraints of any constraint type in order to meet all requirements raised by data practitioners:

4.2 *How to represent constraints of any constraint type and how to reduce the representation of constraints of any constraint type to the absolute minimum?*

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

4.3 *How to ensure for any constraint type that RDF data is consistently validated against semantically equivalent constraints of the same constraint type across RDF-based constraint languages?*

4.4 *How to ensure for any constraint type that semantically equivalent constraints of the same constraint type can be transformed from one RDF-based constraint language to another?*

Even though SPIN provides a convenient way to represent constraint violations and validate data against constraints, the implementation of a high-level constraint language still requires a tedious mapping to SPIN with a certain degree of freedom how constraints of a certain type are checked and how violations of constraints of a particular type are represented. Checks of semantically equivalent constraints of the same type should detect the same set of violations regardless of the used language. This means that whenever semantically equivalent constraints in different languages are checked on RDF data they should point out the same violations.

As there is no standard way to define constraints, semantically equivalent constraints of the same type are expressible by a variety of constraint languages - each of them different in syntax and semantics. Transformations of semantically equivalent constraints of the same type from one language to another are important (1) to enhance the interoperability of constraint languages, (2) to resolve misunderstandings and ambiguities in interpretations of constraints, and (3) to avoid the necessity to understand several languages.

Furthermore, by providing just one SPIN mapping for each constraint type independently of concrete languages, we simplify implementing constraint types for both existing and newly developed constraint languages.

Research Question 5 *What is the role reasoning plays in practical data validation and for which constraint types reasoning may be performed prior to validation to enhance data quality?*

The set of constraint types forms the basis to investigate the role that reasoning and different semantics play 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.

We examine the effect of reasoning to the validation process for each constraint type, i.e., we investigate for each constraint type if reasoning may be performed prior to validation to enhance data quality either by resolving violations or by raising valuable violations and solving them.

We furthermore examine the effects of reasoning on the performance of constraint types. Therefore, we investigate for each constraint type how efficient in terms of runtime validation is performed with and without reasoning.

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.

We use these findings to determine which reasoning constraint types the most common constraint languages enable to express and give directions for the further development of constraint languages by revealing which languages do not cover certain constraint types for which reasoning can be done prior to validation to improve data quality.

1.3 Acknowledgements, Publications, and Research Data

Parts of this thesis result from joint work within four international working groups to which we actively contribute. At the end of each chapter, we describe on which publications and efforts the chapter is based on and clearly differentiate between individual and collaborative endeavors within these working groups.

Main aspects of this thesis have been published in form of 30 research papers (6 journal articles, 9 articles in conference proceedings, 3 articles in workshop proceedings, 2 specifications, and 10 technical reports). For all (except one) journal articles, conference articles, and workshop articles, the author of this thesis is the first author and 12 papers were presented at international conferences and workshops. In a technical report serving as the appendix of this thesis [135], we list the references to these publications and group them by chapter and publication type. Please note that in 2015, the last name of the author of this thesis changed from *Bosch* to *Hartmann*.

To make the research results of this thesis available in a sustainable way, all research results together with direct links to them have been published at the *Karlsruhe Institute of Technology (KIT)* [136] and 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.

2

Foundations for RDF Validation

In this chapter, we lay the ground for the subsequent chapters of this thesis and outline the results of previous research in the fields of RDF and XML validation.

As it is similar to our effort collaboratively collecting RDF validation related case studies, use cases, requirements, and solutions in a comprehensive and structured way, we evaluate *social requirements engineering* regarding a possible reuse, i.e., the use of social software to support collaborative requirements engineering (see Section 2.1).

In Section 2.2, we explain (1) what a schema language for XML is, (2) how *XML validation* is defined, and (3) in form of a complete and intuitive running example which features the major schema languages DTD, XML Schema, RELAX NG, Schematron, and the proof of concept schema language Examplotron possess.

To be able to directly validate XML using common RDF validation tools 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 XML Schemas and conforming XML documents into OWL ontologies and corresponding RDF data. Section 2.3 serves to lay the theoretical background for meta-model based model transformations by (1) comprehensively comparing XML/XML Schema with RDF/OWL, (2) giving an introduction to meta-modeling by means of the Meta Object Facility, and (3) providing an in-depth overview of existing approaches translating XML/XML Schema into RDF/OWL and comparing them in detail with the suggested approach.

We generalize from uni-directional transformations of XML Schema models into OWL models to bidirectional transformations between models of any meta-model such as OWL, XML Schema, relational database schemas, Java, and UML. Meta-model based transformations of arbitrary models to OWL models enable to convert any data to RDF and to validate any data accord-

ing to constraints extractable from models of arbitrary meta-models using common RDF validation tools.

In Section 2.4, we (1) define the term *RDF validation*, (2) introduce current languages for the formulation of constraints and the validation of RDF data against these constraints, and (3) give a preview of the constraint language currently developed by the W3C working group.

Finally, we (1) define what semantics is and provide intuitive examples for each semantic assumption, (2) compare the underlying semantics for RDF and XML validation on the one side and OWL reasoning on the other side, and (3) explain why OWL can be considered as a constraint language under the same semantics adopted when validating RDF data (see Section 2.5).

2.1 Social Requirements Engineering

We collected the findings of the DCMI and W3C working groups on RDF validation and initiated a community-driven publicly available database with the intention to collaboratively collect case studies, use cases, requirements, and solutions in a comprehensive and structured way (see Chapter 5).

Requirements engineering is recognized as a crucial part of project and software development processes [15, 16, 158, 256]. Similar to our collaborative effort, [209] propose *social requirements engineering*, i.e., the use of social software like wikis to support collaborative requirements engineering. Their approach focuses on simplicity and supports in particular the early phases of requirements engineering with many distributed participants and mainly informal collaboration. They emphasize the social experience of developing requirements for software systems: Stakeholders are enabled to collaboratively collect, discuss, improve, and structure requirements. Under the supervision of experts, the requirements are formulated in natural language and are improved by all participants step by step. Later on, experienced engineers may clean and refine requirements. As basis for their work, they developed a generic approach (*Softwiki*) using semantic technologies and the *SWORE* ontology[1] for capturing requirements relevant information semantically [210].

We evaluated the implementation and the ontology regarding a possible reuse, but it turned out that Softwiki focuses clearly on the requirements within a traditional software development process, while we need a broader view including case studies, use cases, and various implementing solutions. Nevertheless we will reuse parts of the SWORE ontology and include links wherever possible.

[1] The SWORE ontology and a prototypical implementation of the approach is online available at: http://aksw.org/Projects/SoftWiki.html

2.2 XML Validation

In this section, we explain (1) what a schema language for XML is, (2) how XML validation is defined, and (3) in form of a complete and intuitive running example which features the major schema languages DTD, XML Schema, RELAX NG, Schematron, and the proof of concept schema language Examplotron possess. In-depth comparisons of major schema languages are provided by [204, 308]. An exhaustive list of minor schema languages would also include schema languages like (1) the *Document Structure Definition Language (DSDL)*[2] [166], a schema language for XML and SGML to enrich the capabilities of DTD with some datatyping and occurrence features from other schema languages, (2) *xlinkit* [233], a first-order logic-based rule language for checking link consistency, and (3) the minimal schema language *Hook* [172] which is based on partial ordering and which is even more terse than DTD.

All schema languages define transformations applied to a class of XML instance documents. XML Schemas, e.g., should be thought of as transformations taking an XML document as input and generating a validation report which includes at least a return code reporting whether the document is valid. One important consequence of realizing that XML Schemas define transformations is that one should consider general purpose transformation languages like XSLT and APIs as alternatives when choosing a schema language.

An XML document conforming to a particular schema is considered to be *valid*, and the process of checking that conformance is called *validation*. We can differentiate between at least four levels of validation enabled by schema languages: (1) markup or structure validation to check document structures, (2) content or datatype validation to check if individual leaf nodes of an XML tree correspond to certain datatypes, (3) integrity or link validation to check links between nodes within a document or between documents, and (4) arbitrary additional business rules. Validating markup and datatypes are the most powerful validation levels. Link validation, in particular between documents, is poorly covered by current schema languages [308].

2.2.1 Document Type Definition (DTD)

One of the major schema languages is *Document Type Definition (DTD)* [60], a simplified version of the *Standard Generalized Markup Language (SGML)* [160], a meta-language used for defining markup languages. A DTD schema describes the structure of a class of XML documents via element and attribute-list declarations. Element declarations name the allowable set of elements within XML documents of a particular class, and specify whether and how sequences of declared elements and of character data may be contained within each element. Attribute-list declarations name the allowable set of attributes for each declared element, including the types of attribute values and an explicit set of valid attribute values.

[2] http://dsdl.org

Schema languages enable to express the following constraints on XML documents of a certain class: (1) The element *library* must be the root element of XML documents of the specified class of XML documents. (2) The element *library* contains the child elements *book* and *author* in exactly this order (3) The element *library* must have at least one child element *book*. (4) The element *library* may have multiple child elements *author*. (5) The element *library* cannot have any attributes. (6) The element *book* contains the child elements *isbn*, *title*, and *author-ref* in exactly this order (7) The element *book* must exactly have one child element *isbn*. (8) The element *book* must exactly have one child element *title*. (9) The element *book* may have multiple child elements *author-ref*. (10) The element *book* must exactly have one attribute *id*. (11) Values of the attribute *id* of the element *book* must be unique within the XML document (12) The element *author-ref* must exactly have one attribute *id*. (13) The element *author-ref* cannot have any child elements or text content. (14) The element *author* must exactly have one child element *name*. (15) The element *author* must exactly have one attribute *id*. (16) Values of the attribute *id* of the element *author* must be unique within the XML document (17) The element *isbn* can only contain text content. (18) The element *isbn* cannot have any attributes. (19) The element *title* can only contain text content. (20) The element *title* cannot have any attributes. (21) The element *name* can only contain text content. (22) The element *name* cannot have any attributes. Each of these constraints is representable in a DTD schema:

```
DTD:
<!ELEMENT library (book+, author*)>

<!ELEMENT book (isbn, title, author-ref*)>
<!ATTLIST book
  id ID #REQUIRED
>

<!ELEMENT author-ref EMPTY>
<!ATTLIST author-ref
  id IDREF #REQUIRED
>

<!ELEMENT author (name)>
<!ATTLIST author
  id ID #REQUIRED
>

<!ELEMENT isbn (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT name (#PCDATA)>
```

The XML document below satisfies all these constraints represented by major schema languages:

```
XML:
<library>
    <book id="_978-0152023980">
        <isbn>978-0152023980</isbn>
        <title>The Little Prince</title>
        <author-ref id="Antoine-de-Saint-Exupéry"/>
```

```
    </book>
    <author id="Antoine-de-Saint-Exupéry">
        <name>Antoine de Saint-Exupéry</name>
    </author>
</library>
```

2.2.2 XML Schema (XSD)

By replacing DTD as the way in which XML documents are described and validated, *XML Schema (XSD)* [23, 108, 304] was introduced as the primary schema language to specify and constrain the syntactic structure of a set of XML documents and to determine the terminology of particular domains of interest [76, 310, 322].

The main reasons why the W3C developed a new schema language are: (1) DTD does not support namespaces. XSD, in contrary, enables to validate XML documents based on namespaces which allows for the distinction between identical terms being used in different contexts. (2) XSD offers a datatyping system which is richer than the DTD datatyping system. In particular, XSD defines a greater and more complex set of datatypes (such as booleans, numbers, dates, times, and currencies), the datatype system does not only apply to attributes, and users can define their own datatypes [131, 325].

XML-Data, XDR, DCD, SOX, and DDML influenced the W3C XML Schema Working Group when developing the W3C recommendation of XSD. Finally, XSD became the only surviving member of this family of schema languages. *XML-Data* [201] included most of the basic XSD concepts. *XML-Data Reduced (XDR)* [114] served to refine and subset those ideas down to a more manageable size in order to allow faster progress towards adopting a new schema language for XML. *Document Content Description for XML (DCD)* [59], a means for specifying rules covering the structure and content of XML documents, expressed a subset of XML-Data in a way that is consistent with RDF. *Schema for Object-Oriented XML (SOX)* [92] was influenced by OOP language design and included concepts like interface and implementation. The purpose of the *Document Definition Markup Language (DDML)* [58] was to encode the logical (as opposed to physical) content of DTDs in an XML document. DDML made a clear distinction between structures and data.

XSDs basically consist of element declarations, attribute declarations, simple type definitions, and complex type definitions. These declarations and definitions describe the syntactic structure of XML documents, that means which elements, attributes, and types are allowed or required as the content of elements and in which order. An XSD, which incorporates the constraints of the running example, could be:

```
XML Schema:
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="library">
        <xsd:complexType>
```

```
            <xsd:sequence>
                <xsd:element name="book" maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="isbn" type="xsd:string"/>
                            <xsd:element name="title" type="xsd:string"/>
                            <xsd:element name="author-ref"
                                         minOccurs="0" maxOccurs="unbounded">
                                <xsd:complexType>
                                    <xsd:attribute name="id" type="xsd:IDREF"
                                                   use="required"/>
                                </xsd:complexType>
                            </xsd:element>
                        </xsd:sequence>
                        <xsd:attribute name="id" type="xsd:ID" use="required"/>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="author" minOccurs="0" maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="name" type="xsd:string"/>
                        </xsd:sequence>
                        <xsd:attribute name="id" type="xsd:ID" use="required"/>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

2.2.3 Regular Language for XML Next Generation (RELAX NG)

The *Regular Language for XML Next Generation (RELAX NG)*[3] [81] is a grammar-based schema language for XML. The key features of RELAX NG are: (1) it is closer to a description of XML instance documents in ordinary English and simpler than XSD,[4] (2) it is both easy to learn and use for schema creators and easy to implement for software developers, (3) it has both an XML syntax and a compact non-XML syntax, (4) it has a solid theoretical basis, (5) it supports XML namespaces, (6) it treats attributes uniformly with elements, (7) it has unrestricted support for unordered and mixed content, and (8) it can partner with a separate datatyping language such as XSD [311].[5]

RELAX NG is the result of the merger of RELAX and TREX. The *Regular Language Description for XML*[6] *(RELAX)* [164] is both simple and built on a solid mathematical foundation. *Tree Regular Expressions for XML (TREX)*[7] [80] is basically the type system of XDuce with an XML syntax and additional features. *XDuce*[8] [155, 156] is a statically typed programming language that is specifically designed for processing XML data. Although its purpose is not

[3] http://relaxng.org
[4] Article providing an in-depth comparison between RELAX NG and XSD, cf. [309]
[5] Article discussing aspects of the design of RELAX NG including the treatment of attributes, datatyping, mixed content, and unordered content, cf. [79]
[6] http://www.xml.gr.jp/relax
[7] http://thaiopensource.com/trex
[8] http://xduce.sourceforge.net

to be a schema language, its typing system has influenced the further development of schema languages. The primary concept in TREX is the pattern. A TREX pattern, which is itself an XML document, specifies a pattern for the structure and content of an XML document to identify a class of XML documents consisting of those documents that match the pattern. A RELAX NG schema including the constraints of the running example could be:

```
RELAX NG:
<grammar
  xmlns="http://relaxng.org/ns/structure/1.0"
  datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
    <start><choice><ref name="library"/></choice></start>
    <define name="library">
        <element name="library">
            <oneOrMore><ref name="book"/></oneOrMore>
            <zeroOrMore><ref name="author"/></zeroOrMore>
        </element>
    </define>
    <define name="book">
        <element name="book">
            <ref name="id-attribute"/>
            <ref name="isbn"/>
            <ref name="title"/>
            <zeroOrMore>
                <element name="author-ref">
                    <attribute name="id"><data type="IDREF"/></attribute>
                    <empty/>
                </element>
            </zeroOrMore>
        </element>
    </define>
    <define name="id-attribute" >
        <attribute name="id"><data type="ID"/></attribute>
    </define>
    <define name="isbn">
        <element name="isbn"><text/></element>
    </define>
    <define name="title">
        <element name="title"><text/></element>
    </define>
    <define name="author">
        <element name="author">
            <attribute name="id"><data type="ID"/></attribute>
            <element name="name"><text/></element>
        </element>
    </define>
    <define name="name">
        <element name="name"><text/></element>
    </define>
</grammar>
```

2.2.4 Schematron

Schematron is a schema language offering an expressive power that DTD, XSD, and RELAX NG can't match. If rules have to be checked that go beyond checking the pure syntactic structure of XML documents, so-called secondary-level validation tools like *Schematron*[9] [167] have to be introduced to overcome

[9] http://www.schematron.com
http://xml.ascc.net/resource/schematron/schematron.html

these limitations when validating XML [312]. Examples of constraints which cannot be specified by means of DTD, XSD, or RELAX NG are: (1) If an element has a specific value as content, then certain child elements must be present. (2) A start date must be earlier than or equal to an end date. (3) If an element has attribute *a*, it must also have attribute *b*. (4) If an element has a parent *a*, then it must have an attribute *b*.[10]

Most schema languages for XML rely on regular grammars when defining constraints, a fundamental paradigm in the design of these languages. Schematron, on the other hand, is an XPath/XSLT-based schema language for defining context dependent validation rules and validating XML documents according to them. Schematron relies almost entirely on XPath query patterns for defining these rules. To write a full schema with Schematron, a schema designer needs to take care to include all the rules required to qualify the structure of a particular set of XML documents [253].

A unique feature of Schematron is its user-centric approach which enables to raise user-friendly error messages rather than standard error messages generated by a schema processor. This allows to document individual patterns in a schema and to give very direct feedback to users.

Schematron doesn't directly support structure or datatype validation, but a schema author may write rules which implement these structure and datatype checks. Even though the features of schema languages for XML are more complementary than overlapping, there is room for interesting combinations with Schematron. The combination of XSD and Schematron enables the use of each language for the purpose for which it has been designed: structure and datatype validation for XSD and defining rules for Schematron. A possible combination of RELAX NG, XSD, and Schematron would be to use RELAX NG for validating the structure, XSD for validating datatypes, and Schematron for formulating rules [241].

The partial Schematron schema below shows the power of Schematron to define additional constraints not expressible by DTD, XSD, or RELAX NG, such as to check if values of the attribute *id* of the element *book* are derived from the text content of its child element *isbn*:

```
Schematron:
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
    <sch:pattern>
        <sch:rule context="/">
            <sch:assert test="library">library must be the root element</sch:assert>
        </sch:rule>
        <sch:rule context="/library">
            <sch:assert test="book">at least 1 book</sch:assert>
            <sch:assert test="not(@*)">library cannot have any attributes</sch:assert>
        </sch:rule>
        <sch:rule context="/library/book">
            <sch:assert test="not(following-sibling::book/@id=@id)">
                book ids must be unique within the XML document
            </sch:assert>
```

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

```
            <sch:assert test="@id=concat('_', isbn)">
                book ids must be derived from the text content of isbn
            </sch:assert>
        </sch:rule>
        <sch:rule context="/library/*">
            <sch:assert test="name()='book' or name()='author'">
                allowed child elements of library: book, author
            </sch:assert>
        </sch:rule>
    </sch:pattern>
</sch:schema>
```

2.2.5 Examplotron

Classical schema languages such as DTD, XSD, RELAX NG, or Schematron rely on a modeling of either the structure (and eventually the datatypes) that a document must follow to be considered as valid or on rules that need to be checked. This modeling relies on specific XML serialization syntaxes, needed to be understood and very different from the instance documents.

Starting from the observation that (1) XML documents are usually much easier to understand than their schemas describing and validating them and that (2) schema languages often need to give examples of XML documents to help human readers to understand their syntax, Examplotron has been proposed.

Examplotron [313] is a proof of concept to define a lightweight schema language that is based on XML example trees. Examplotron uses document instances to generate a schema, a document with all the elements and attributes that are required plus some additional elements to specify optionality [171]. An XML document to derive an appropriate schema covering the running example could be:

```
Examplotron:
<library xmlns:eg="http://examplotron.org/0/" eg:assert="not(@*)">
    <book id="_978-0152023980"
        eg:occurs="+"
        eg:assert="not(following-sibling::book/@id=@id) and @id=concat('_', isbn)">
        <isbn eg:assert="not(@*)">978-0152023980</isbn>
        <title eg:assert="not(@*)">The Little Prince</title>
        <author-ref id="Antoine-de-Saint-Exupéry" eg:occurs="*"/>
    </book>
    <author id="Antoine-de-Saint-Exupéry"
        eg:occurs="*"
        eg:assert="not(following-sibling::author/@id=@id)">
        <name eg:assert="not(@*)">Antoine de Saint-Exupéry</name>
    </author>
</library>
```

2.3 RDFication of XML

In order to be able to directly validate XML data using common RDF validation tools against semantically rich OWL axioms when using them in terms

of constraints and extracting them from XSDs adequately representing particular domains, we propose on formal logics and the XSD meta-model based automatic transformations of XSDs and conforming XML documents into OWL ontologies and corresponding RDF data (see Chapter 4).

In this section, we lay the theoretical background for meta-model based model transformations. First, we comprehensively compare XML/XSD and RDF/OWL by examining their similarities and differences with regard to their initial intentions, widespread usage, modeling goals, underlying semantics, and structuring data. Second, we give an introduction to meta-modeling using the Meta Object Facility as fundamental component of the Model Driven Architecture to support the model-driven engineering of software systems. And third, we provide an in-depth overview of existing approaches translating XML/XSD into RDF/OWL, and compare them in detail with the suggested approach.

2.3.1 XML and XML Schema Compared to RDF and OWL

Although the first official concrete syntax of RDF was XML-based [117], RDF and XML have been developed separately, which has led to different modeling foundations and which is the reason why RDF and XML serve different purposes. XSD and OWL are both extensively used to determine the vocabulary of particular domains and to describe conceptual models about data of various domains, even though they follow different modeling goals. XSD and OWL have in common that they are similar to object models, i.e., they have an object-oriented foundation. For instance, XSD and OWL both have the notion of class hierarchy and specialization.

More effective and efficient cooperations between individuals and organizations are possible if they take advantage of the synergies of both modeling languages, i.e., if they agree on a common syntax specified by XSDs and if they have a common understanding of the domain classes and their relations defined using OWL.

Similarities and Differences Regarding Widespread Usage, Initial Intention, and Modeling Goals

The *Extensible Markup Language (XML)* [60] is extensively used to (1) represent a large set of information, (2) exchange data in distributed environments, and (3) integrate data of various sources and domains like the B2B [72], the multimedia [118], and the biomedical domain [251, 282]. XML became an important technology in many biomedical domains, driven primarily by the desire for greater interoperability between biomedical software applications [282]. Thereby, XML is used to define information models that describe the structure and the content of biomedical data that can be exchanged between applications [251].

XML is a language that defines a generic syntax to store and exchange documents by means of a tree-based structure. The XML data model describes the terminology and the syntactic structure of node labeled trees [37]. XML is document-oriented which means that XML is intended to structure and exchange documents, but is used data-oriented, i.e., to structure and exchange data - a purpose for which it has not been developed.

An XML document may be an instance of an *XML Schema (XSD)* [108, 304] - the primary, widely adopted, and mostly used technology for defining structural constraints on sets of XML documents and validating XML documents on these constraints. XSDs determine the terminology of particular domains and the syntactic structure of XML documents of specific classes. XSDs are used to define integrity constraints for documents and semi-structured data [181]. The XSD construct *xsd:choice*, for example, is used to define an exclusive or of XML elements.

Just like XML, the initial intention of XSD concentrates on structuring documents instead of structuring data, but is generally used to structure data in XML documents. XSDs basically consist of element declarations, attribute declarations, simple type definitions, and complex type definitions. These declarations and definitions describe the structure of XML documents of certain classes, i.e., which elements, attributes, and types are allowed or required as the content of elements and in which order.

The *Resource Description Framework (RDF)* [89, 143, 280] is the standard data format of the Semantic Web. RDF data is published in the increasingly popular and widely adopted LOD cloud[11] [278] to get linked with a huge number of RDF data sets of different topical domains.[12] As RDF is an established standard, there is a plethora of tools which can be used to interoperate with data represented in RDF which may semantically conform to RDFS/OWL ontologies.

RDF Schema (RDFS) [61] provides a data-modeling vocabulary for RDF data as extension of the basic RDF vocabulary. RDFS provides mechanisms for describing groups of related resources and the relationships between these resources. Resources again are used to determine characteristics of other resources, such as the domains and ranges of properties. The RDFS class and property system is similar to the type systems of object-oriented programming languages such as Java.

The *Web Ontology Language (OWL)* [27, 151] in its current version 2 is an expressive language used to formally specify the semantics of conceptual models about data and therefore enables software to understand and properly process data according to its intended semantics. OWL is used to specify semantic information about domains, to describe relations between domain classes, and thus allows the sharing of conceptualizations. OWL has become a popular standard for data representation, data exchange, and data integration

[11] http://lod-cloud.net
[12] http://linkeddata.org

of heterogeneous data sources and provides facilities for developing very powerful reasoning services enabling to derive implicit knowledge out of explicitly stated knowledge. OWL is based on formal logic and on the subject-predicate-object triples from RDF.

Differences Regarding Semantics

We distinguish the syntactic level of XSD and the semantic level of OWL. XSD is used to define the syntactic structure of sets of XML documents rather than their underlying model. The minimum or maximum occurrence of an element, e.g., describes what can appear at that lexical position and is quite different to an OWL cardinality restriction. OWL, in contrast, is used to formally and semantically describe domains and to model semantic relationships between domain classes. [99] even assume, that a suitable automatic mapping between XML and RDF is impossible, since XML does not contain any semantic constraints. It is claimed that XML represents the document structure, but does not contain any information about the meaning of the content. Other approaches, however, assume that there is some semantics in XML documents, which can be discovered out of the document structure. [216], for instance, assumes that every XML document has an RDF model and therefore tries to detect semantics in XML instance documents and to map them to RDF.

Even though it was not the initial intention of XSD to define semantics of data models, XSDs are commonly used to represent semantic information like OWL, but to a lesser extend. The difference is that OWL provides much more powerful features for representing semantic information and that OWL is used to specify semantics of domain data models both formally and explicitly. Semantics of XSDs, on the other hand, are only defined implicitly so that different XML documents have the same implicit semantics in case they correspond to the same XSD. Examples of implicit XSD semantics are class memberships of XML elements and relationships between XML elements. For transformations of XML/XSD into RDF/OWL, this means that implicit semantics of XSD constructs must be made explicit and based on formal semantics.

XSD and OWL are founded on varying underlying semantics. XSD is based on the *unique name assumption (UNA)*, i.e., two resources with different names are indeed different, whereas OWL is not based on the UNA, but on the *non-unique name assumption (nUNA)*, i.e., two resources with different names may be the same and therefore may be explicitly declared being different (*owl:differentFrom*) or identical (*owl:sameAs*).

XSD is based on the *Closed-World Assumption (CWA)* and OWL on the *Open-World Assumption (OWA)*. According to the OWA, missing information is undecided, and CWA means that missing information is considered to be false. We refer to Section 2.5 for an in-depth comparison between the semantics for (1) XML validation, (2) RDF validation, and (3) OWL reasoning.

Differences Regarding Structuring Data

XML and RDF are different in how they structure data. XML is based on a tree model where only nodes are labeled and outgoing edges are ordered. This model originates from semi-structured data and databases. The XSD construct *sequence* is used to specify the explicit order of XML elements contained in parent elements.

RDF, in contrast, is set-oriented and based on a directed graph model where edges have labels, but are unordered. RDF distinguishes between resources (e.g., books and persons) and properties (e.g., has author) while XML does not, i.e., both resources and properties would be elements. This model originates from knowledge representation languages such as Frames [225] and Description Logics [12, 13, 196, 272]. These structural differences of the data models, however, represent no obstacle, as trees are a specialization of graphs.

Differences Regarding Reasoning

The underlying *proof-theory* of OWL enables inference engines, also called reasoners, to execute automatic inferences in order to derive implicit triples out of explicitly stated triples. XSD does not offer reasoning services as XSDs are only used to validate XML elements and XML attributes which are explicitly stated in conforming XML documents.

Differences Regarding Global Uniqueness of Identifiers

OWL and XSD differ with regard to the global uniqueness of defined names. OWL IRIs (Internationalized Resource Identifiers) [105] are globally unique, whereas, XSD global element declarations and global complex type definitions are not globally unique but unique within namespaces. For software engineering languages like UML, package names must be globally unique, class names have to be unique within a package, and attribute names must be unique within a class.

2.3.2 Model Driven Architecture

Model-Driven Engineering (MDE) [73] is a methodology to develop systems using models which is seen as an emerging solution to handle complex and evolving software systems. The *Model Driven Architecture (MDA)* [113, 215] is an initiative of the standardization organization *Object Management Group (OMG)*[13] to support the model-driven engineering of software systems which is based on the conviction that modeling gives a good foundation to develop and maintain software systems. MDA uses modeling to raise the abstraction level and to manage the complexity of systems.

[13] http://www.omg.org

The *Meta Object Facility (MOF)* [247] is a fundamental component of the MDA. To support the MDA initiative, it is essential that designed models are commonly understood by all involved parties. This requires the ability to specify the meaning of models by means of meta-models. A *meta-model* specifies the constructs of a *modeling language* which is used to formulate models.

MOF is organized in four layers to adequately structure the relationships between a meta-model and *model instances* that represent the objects in the real world. These four layers are the information (M0), the model (M1), the meta-model (M2), and the meta-meta-model (M3) layers (see Figure 2.1 for a simplified representation of the MOF layers).

The undermost *information layer (M0)* of this hierarchy contains real world objects like the book *A Study in Scarlet* whose author is *Arthur Conan Doyle*. The *model layer (M1)* contains the definition of required structures like concrete properties and classes used to classify information. In our example, the classes *Book* and *Person* and the property *author* are defined. If these structures are combined, they describe the *model* for a given domain.

The *meta-model layer (M2)* specifies the terms in which the model is expressed. In our example, we would state that models are expressed by *classes* and *properties* by instantiating appropriate *meta-classes*. The *meta-meta-model layer (M3)* defines the modeling constructs that can be used to define meta-models on the meta-model level. The meta-meta-model layer contains only the simplest set of concepts which are used to define the structure and the semantics of any meta-model.

2.3.3 Approaches Transforming XML/XSD into RDF/OWL

In this section we provide an in-depth overview of existing approaches converting XML/XSD into RDF/OWL, and compare them in detail with the suggested approach. The main differences are:

- **Meta-model based transformations.** In comparison to previous tools, the novelty of the devised approach is that the translation of XSDs into OWL ontologies is based on the XSD meta-model on level M2.

- **Transformations on levels M0 and M1.** The majority of the tools are designed to transform either XML into RDF on the assertional knowledge level M0 or schemas into OWL on the terminological knowledge level M1. We follow a complete approach converting XML documents to OWL individuals (M0) and XSDs into OWL ontologies (M1) which correspond to an OWL ontology representing the XSD meta-model (M2).

- **Automatic transformations.** Many existing approaches propose to map XML to RDF (M0) and/or XSD to RDFS/OWL (M1) in a manual manner. The overall transformation process of the suggested approach,

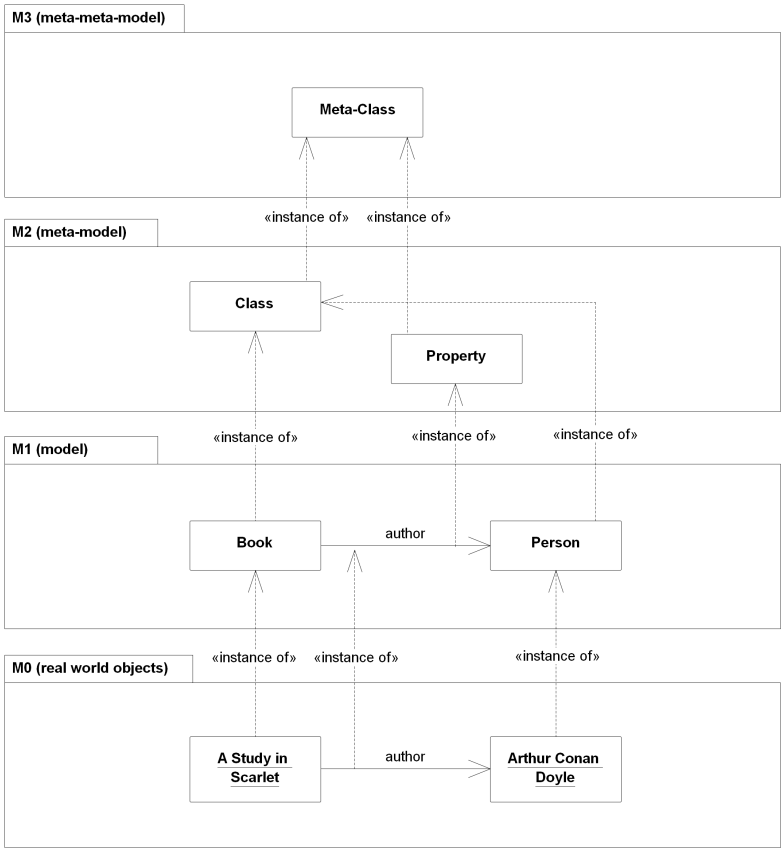

Fig. 2.1. Meta Object Facility Layers

however, is performed semi-automatically. (1) We translate XSDs and XML into OWL ontologies and their RDF representation in an automatic way. Generated OWL ontologies can be used directly after they have been created but they might not be as useful as manually created domain ontologies. (2) In order to automatically derive OWL domain ontologies out of these generated OWL ontologies, we have to define SWRL rules by hand which can then be executed by rule engines.

- **High expressivity of OWL 2 ontologies.** Divers existing tools generate RDFS ontologies and not the more expressive OWL or even OWL 2 ontologies. The suggested approach converts XSDs to OWL 2 ontologies as we want to use the high expressive power of OWL 2 to adequately express the formal semantics of the resulting ontologies.

Several strategies lifting the syntactic level of XML documents and XSDs to the semantic level of RDF and OWL ontologies can be distinguished. We

cluster these transformations tools into three classes. The classification depends on the level of conversion which happens either on the instance (M0), the conceptual (M1), or both, the instance and the conceptual level.

Transformations on Instance Level M0

On the instance level M0, [180] developed the so-called *RDF Schema mapping ontology* which enables a one-way mapping of XML documents to RDF. Relevant content of XML documents can be identified. As extension to this approach, [17] introduced a bidirectional mapping of XML components to RDF. The *WEESA* system implements an automatic transformation from XML to RDF using an OWL ontology which is created manually from corresponding XSDs and manually defined rules. XML document instances, however, are not mapped to OWL individuals which correspond to this OWL ontology [266]. [251] transform XML documents to individuals of an OWL ontology describing the serialization of the XML document. SWRL is used to map these instances to individuals of a domain ontology.

Transformations on Schema Level M1

On the conceptual level M1, we distinguish between approaches converting *schema languages* to RDFS or to OWL. Several languages for writing schemas like *Document Type Declaration (DTD)* [60], XSD [304], *Document Structure Description (DSD)* [179] and Relax NG [81] exist. The prototype *OntoLiFT* [317] offers a generic means for converting schemas expressed by arbitrary schema languages to RDFS ontologies semi-automatically. In a first step, schemas are transformed into regular tree grammars consisting of non-terminals, terminals, start symbols and production rules [231]. In a second step, non-terminals and terminals are converted to RDFS classes and production rules are mapped to RDF properties. [6] base transformations of multiple models such as OWL, XSD, and RDF on a meta-model, the *superimposed meta-model*. The limitation is that the meta-models of XSD and OWL are not complete.

Transformations on Instance Level M0 and Schema Level M1

On the instance level M0 and on the conceptual level M1, there are methods transforming XML to RDF and XSD to either RDFS or OWL. Within the EU-funded project called *Harmonise*, the interoperability of existing standards for the exchange of tourism data has been achieved by transformations of XML documents and XSDs into RDF and RDFS ontologies which are mapped to each other [100]. [252] transform XML document instances into OWL ontologies, even though associated XSDs do not exist. As a consequence, unstructured contents can be converted to OWL ontologies as well. XSDs can also be mapped to OWL ontologies as XSD documents are represented in

XML, too. It is possible to generate new OWL ontologies from scratch and to extend existing ones. They evolved *XML Master*, a language to describe OWL ontologies declaratively. XML Master combines the Manchester OWL Syntax [152] and XPath to refer to XML content. They criticize the limited and unsatisfactory number of OWL constructs supported by current tools converting XSDs to OWL ontologies. Thus, they convert all OWL constructs. One shortcoming is that the mapping language expressions have to be written manually and therefore XML documents and XSDs cannot be transformed into OWL ontologies completely automatically. Another drawback is that ontology engineers have to be familiar with the Manchester OWL Syntax and XPath to express the mappings. [109] propose both mappings from XML to RDF (M0) and XSD to OWL (M1) which are independent from each other. This means, OWL individuals do not necessarily correspond to the OWL ontology, since declarations and definitions of XML documents may be transferred to different OWL constructs. [188] transfer XSD components to OWL language constructs at the terminological level (M1) and XML document instances to OWL individuals at the assertional level (M0). Thereby, XPath expressions are applied to select the content of XML documents. Besides that, the approach of [305] is very similar to the method of [188].

[32] devised mappings between XML and RDF (M0) and between XSD and OWL (M1). They assume that XML documents are structured like relational databases which is the reason why relational structures of XML documents are discovered and represented in OWL. Relations correspond to classes, columns to properties, and rows to instances. XML data model elements are mapped automatically to components of the OWL data model. Named simple and complex types, for instance, are transferred to classes. Elements, containing other elements or having at least one attribute, are converted to classes and object properties between these classes. Both, elements, including neither attributes nor sub-elements, and attributes, which are assumed to represent database columns, are transformed into data properties with the surrounding element as domain. Besides, XML cardinality constraints are transformed into equivalent OWL cardinality restrictions.

2.3.4 Meta-Model Based Model Transformations

We generalize from uni-directional transformations of XSD models into OWL models to bidirectional transformations between models of any meta-model like XSD [304], RDFS [61], OWL 2 [27], relational database schemas, Java [126], the Unified Modeling Language (UML) 2 [243, 244], the Systems Modeling Language (SysML) [245], and the Business Process Model and Notation (BPMN) 2 [246]. The Common Warehouse Meta-Model (CWM) [242] may be used as meta-model for relational database schemas. The main purpose of CWM is to enable an easy interchange of warehouse and business intelligence metadata between warehouse tools, platforms, and metadata repositories in distributed heterogeneous environments. Meta-model based transformations of arbitrary models to OWL models enable to convert any data

to RDF and to validate any data according to constraints extractable from models of arbitrary meta-models using common RDF validation tools.

Query/View/Transformations (QVT) [250] is a language to define transformations between models on the meta-model level. QVT is based on MOF and the *Object Constraint Language (OCL)* [248]. OCL is a formal, declarative, and precise language for describing rules which provides constraint and object query expressions on any MOF model or meta-model that cannot otherwise be expressed by diagrammatic notation. QVT can be used to define transformation models describing transformations between source and target models of any meta-model. Instead of transforming models directly on the model level, model transformations are specified on the meta-model level using meta-model constructs and semantics. By defining transformations on a higher meta-level, model transformations do not depend on the converted models. They only depend on the involved meta-models which enables an elegant description of the transformation.

The *Eclipse Modeling Project*[14] focuses on model-based development technologies by providing a set of modeling frameworks, tooling, and implementations of standards. As core of the Eclipse Modeling Project, the *Eclipse Modeling Framework (EMF)* [293] provides a basic framework for the model-driven engineering of software systems which supports MOF-based meta-modeling. As part of EMF, *ecore* [71] has emerged as the de facto standard for the definition of meta-models and therefore serves as meta-meta-model to specify meta-models. Ecore is similar to *Essential MOF (EMOF)* [247], the lightweight core of MOF.

Prerequisite for performing meta-model based model transformations is to represent meta-models and models as direct and indirect instances of the ecore meta-meta-model. There are multiple MOF-based meta-models which are already represented conforming to ecore, such as OWL 2,[15] XSD,[16] UML 2,[17] and BPMN 2.[18] Additionally, [63] provides MOF-based meta-models for OWL ontologies [64, 65, 69, 70], OWL ontology mappings [68], SWRL [66, 67, 154], and F-Logic [177, 178] which integrates logic with object-oriented programming. Ecore represetations of meta-models and models are physically represented by the standard exchange format *XML Metadata Interchange (XMI)* [249].

As model transformations are defined by means of the abstract syntax instead of concrete syntaxes of a language, they become independent of any particular representation and transformations do not have to be defined for each concrete syntax. OWL 2, for instance, has many concrete syntaxes such as the Functional-Style syntax [27], the RDF 1.1 Turtle syntax [262], the RDF

[14] https://eclipse.org/modeling
[15] https://www.w3.org/2007/OWL/wiki/MOF-Based_Metamodel
[16] Directly integrated in EMF as Eclipse plug-in
[17] https://projects.eclipse.org/projects/modeling.mdt.uml2
[18] http://wiki.eclipse.org/MDT-BPMN2

1.1 XML syntax [117], the OWL 2 XML serialization [230], the Manchester syntax [152], and JSON-LD 1.0 [289].

There are several independent implementations of QVT. One of the three domain specific languages of QVT is *QVT Operational Mappings (QVTo)* [250], an imperative language that looks similar to other imperative languages such as Java and whose syntax corresponds to the syntax of OCL since QVTo is based on OCL. QVTo is the QVT language with currently the best tool support and the largest fostering community. The black box mechanism of QVTo allows to code complex algorithms in any supported programming language. This is very useful because some algorithms are very hard to formulate in OCL.

2.4 RDF Validation

According to the W3C RDF Data Shapes Working Group, the process of checking conformance of nodes in an RDF graph to some kind of schema is referred to as *validation*. The output of a constraint validation process is a set of validation results, a validation report which indicates whether or not the validated graph conforms to the schema. If any constraint in the schema is not satisfied, the validation report includes one or more constraint violations indicating the source of the violation as well as useful human-readable messages [186].

In this section, we introduce current languages for the formulation of constraints and the validation of RDF data against these constraints. SPARQL, SPIN, OWL, ShEx, ReSh, and DSP are the six most promising and mostly used constraint languages. In addition, the W3C working group currently develops SHACL, an RDF vocabulary for describing RDF graph structures, and OWL reasoning techniques, i.e., terminological and assertional OWL queries, can be executed to determine if data models are consistent, and finally, to prevent poor quality of data models.

Besides the pure formulation of constraints on RDF data, SPIN (open source API), Stardog ICV (as part of the Stardog RDF database), DQTP (tests), Pellet ICV (as extension of the Pellet OWL 2 DL reasoner), and ShEx offer executable validation environments using SPARQL as a language to actually implement RDF data validation.

2.4.1 SPARQL Query Language for RDF (SPARQL)

The *SPARQL Query Language for RDF* [8, 134] is generally seen as the method of choice to validate RDF data according to certain constraints [115], although SPARQL queries can be quite long and complex. SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or just viewed as RDF via middleware. The results of SPARQL queries can be results sets or RDF graphs.

SPARQL is very expressive and can even be used to validate numerical and statistical computations [121]. On the other hand, SPARQL does not allow to support recursive constraints. Hence, [268] recently proposed an extension operator to SPARQL to include recursion. Nevertheless, most SPARQL engines already have functions that go beyond the official SPARQL 1.1 specification.

SPARQL enables to express constraints of high complexity. For assessing the quality of thesauri, e.g., we concentrate on their graph-based structure and apply graph- and network-analysis techniques. An example of such constraints of high complexity is to prevent that a thesaurus contains 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:

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

There have been other proposals using SPARQL combined with other technologies. [284], e.g., propose a combination of SPARQL queries, a set of property paths, and value constraints. [110] propose *RDF Data Descriptions*, a domain-specific constraint language that is compiled into SPARQL queries for constraint checking. When translating relational data in conventional relational databases into RDF, [200] show that SPARQL can be used as a constraint language, akin to SQL in the relational context.

2.4.2 SPARQL Inferencing Notation (SPIN)

In contrast to SPARQL, high-level constraint languages are comparatively easy to understand and constraints can be formulated more concisely. Declarative constraint languages may be placed on top of SPARQL and SPIN when using them as low-level implementation languages. The *SPARQL Inferencing Notation (SPIN)*[19] [185] provides a vocabulary to represent SPARQL queries as RDF triples [184] and uses SPARQL to specify logical constraints and inference rules [115].

The *SPIN SPARQL Syntax* provides a means for representing SPARQL queries in RDF. Having a triple-based SPARQL representation makes it easier to consistently store SPARQL queries together with the data to be validated as well as the conceptual models of domains, i.e., RDFS/OWL definitions [184]. The *SPIN Modeling Vocabulary* is a lightweight collection of properties and classes to support the use of SPARQL to specify logical constraints and inference rules. Based on an RDF representation of SPARQL queries, SPIN

[19] http://spinrdf.org

defines three class description properties: (1) *spin:constraint* to define conditions that all members of a given class must fulfill, (2) *spin:rule* to specify inference rules using SPARQL CONSTRUCT and DELETE/INSERT queries and (3) *spin:constructor* to initialize new instances with default values [183].

SPIN provides a mechanism to encapsulate SPARQL queries into reusable building blocks so that they can be shared on the Semantic Web. In particular, SPIN makes it possible to define new SPARQL functions and query templates together with definitions of their arguments. The *SPIN Standard Modules Library* is a collection of such SPIN functions and templates that are of general use. Among others, this library covers some common domain modeling patterns such as cardinality and value constraints. The SPIN Standard Modules Library also declares standard SPARQL functions like != and BOUND [182].

We make use of SPIN to actually implement existential quantifications in OWL which enforce that instances of given classes must have some property relation to individuals/literals of a certain class or datatype via particular object or data properties:

```
SPIN:
owl2spin:ObjectSomeValuesFrom
  a spin:ConstructTemplate;
  spin:body [
#   CONSTRUCT {
#    _:constraintViolation
#       a spin:ConstraintViolation ;
#       spin:violationRoot ?subject ;
#       rdfs:label ?violationMessage ;
#       spin:violationSource "Existential Quantification" ;
#       :severityLevel :error ;
#       spin:violationPath ?OPE ;
#       spin:fix ?violationFix }
    a sp:Construct ;
    sp:templates (
      [ sp:subject _:constraintViolation ;
        sp:predicate rdf:type ;
        sp:object spin:ConstraintViolation ]
      ... ) ;
#   WHERE {
#     ?subject rdf:type ?subC .
#     ?subC rdfs:subClassOf* ?C .
    sp:where (
      [ sp:subject [ sp:varName "subject" ] ;
        sp:predicate rdf:type ;
        sp:object [ sp:varName "subC" ] ]
      [ a sp:TriplePath;
        sp:subject [ sp:varName "subC" ] ;
        sp:path [
          a sp:ModPath ;
          sp:modMax -2 ;
          sp:modMin 0 ;
          sp:subPath rdfs:subClassOf ] ;
          sp:object [ sp:varName "C" ] ]
#     ?C a owl:Restriction .
#     ?C owl:onProperty ?OPE .
#     ?C owl:someValuesFrom ?CE .
      [ sp:subject [ sp:varName "C" ] ;
        sp:predicate rdf:type ;
        sp:object owl:Restriction ]
      ...
#     FILTER ( ! ( spl:hasValueOfType ( ?subject, ?OPE, ?CE ) ) ) .
```

```
      [ a sp:Filter ;
        sp:expression [
          a sp:not ;
          p:arg1 [
            a spl:hasValueOfType ;
            sp:arg1 [ sp:varName "subject" ] ;
            sp:arg2 [ sp:varName "OPE" ] ;
            sp:arg3 [ sp:varName "CE" ] ] ] ]
#     BIND ( ( ... ) AS ?violationMessage ) .
#     BIND ( ( ... ) AS ?violationFix ) . }
      ... ) ].
```

2.4.3 Data Quality Test Patterns (DQTP)

Inspired by test-driven software development, *RDFUnit* [192] and *Databugger* [191] are test-driven validation frameworks that can run manually and automatically generated test cases against SPARQL endpoints. All test cases are executed as SPARQL queries using a pattern-based transformation approach. These frameworks are based on 17 data quality integrity constraints represented as SPARQL query templates called *Data Quality Test Patterns (DQTP)* [192].

A common type of constraints is to restrict data properties to have a minimum, maximum, or exact number of relationships to literals with selected language tags. One of the SPARQL templates provided by [192] can be used, e.g., to ensure that no language is used more than once per property. Thereby, *P1* is replaced by the property pointing to the literal and *V1* by the language we want to check:

```
DQTP:
SELECT DISTINCT ?s
WHERE { ?s %%P1%% ?c
    BIND ( lang(?c) AS ?l )
    FILTER (isLiteral (?c) && lang(?c) = %%V1%%)}
GROUP BY ?s HAVING COUNT (?l) > 1
```

[101] suggest an incremental and iterative validation workflow for RDF data sets and incorporate the test-driven approach. [190] extend the developed methodology of Linked Data quality assessment and describe how the method is applied in the Natural Language Processing area for assessing NLP data quality.

2.4.4 Web Ontology Language (OWL)

The *Web Ontology Language (OWL)* [27, 151] in its current version 2 is an expressive language used to formally specify the semantics of conceptual models about data and therefore enables software to understand and properly process data according to its intended semantics. OWL is used to specify semantic information about domains, to describe relations between domain classes, and thus allows the sharing of conceptualizations. OWL has become a

popular standard for data representation, data exchange, and data integration of heterogeneous data sources. Besides that, the retrieval of data benefits from semantic knowledge specified in OWL.

OWL for Reasoning

OWL is based on formal logic and the subject-predicate-object triples from RDF. Description Logics (DL) provides the foundational basis for OWL.[20] More specifically, OWL is actually a Description Logic with underlying formal semantics which allows one to assign truth values to syntactic expressions.

In combination with the OWL-based *Semantic Web Rule Language (SWRL)* [154], OWL provides facilities for developing very powerful reasoning services. *Reasoning* on RDF data enables to derive implicit data out of explicitly stated data. RDFS and OWL reasoning enrich query answering over Linked Data by obtaining more complete answers for structured queries over interlinked RDF data sets [257]. *OWL reasoning techniques*, i.e., terminological and assertional OWL queries, can be executed to determine if data models are consistent [294], and finally, to prevent poor quality of data models.

Terminological OWL queries can be divided into checks for global consistency, class consistency, class equivalence, class disjointness, subsumption checking, and ontology classification [294]. A class is inconsistent if it is equivalent to owl:Nothing which indicates a modeling error. If there are not any objects satisfying a class definition, the respective class is not consistent (check for *class consistency*). An ontology is globally consistent if it is devoid of inconsistencies, i.e., the ontology does not have any contradictions (check for *global consistency*). Unsatisfiability is often an indication of errors in class definitions and for this reason the overall quality of ontologies is checked using global consistency checks. By means of *ontology classification*, the ontology's class hierarchy is calculated on the basis of class definitions. *Subsumption checking* on the TBox is used to infer super- and sub-class relations between classes. Applied mutually to every possible pair of classes, subsumption checking makes it possible to build the whole class hierarchy. The fact that an instance is assigned to classes, which are defined to be disjoint, is a signal for poor data quality (*class disjointness*).

Instance checks, class extensions, property checks, and property extensions are classified as *assertional OWL queries* [294]. *Instance checks* are used to test if a specific individual is a member of a given class. The search for all individuals contained in a given class may be performed in terms of *class extensions*. *Property checks* and *property extensions* can be defined similarly with regard to pairs of individuals.

[20] DL can also be used for reasoning with RDF(S), cf. [97]

OWL as Constraint Language for RDF Validation

Validation is not the primary purpose of the design of OWL which has lead to claims that OWL cannot be used for validation. [228] and [229], e.g., discuss the differences between constraints and RDFS/OWL axioms. In practice, however, OWL is well-spread and RDFS/OWL constructs are widely used to tell people and applications about how valid instances should look like. In general, RDF documents follow the syntactic structure and the intended semantics of RDFS/OWL ontologies which could therefore not only be used for reasoning but also for validation.

Universal quantifications in OWL are used to build anonymous classes containing all individuals that are connected by particular properties only to instances/literals of certain classes/data ranges. Hence, universal quantifications can also be used to check, e.g., if all the publications, described in the data to be validated, only have persons as authors:

```
1 OWL 2:
2 :Publication rdfs:subClassOf
3   [ a owl:Restriction ;
4     owl:onProperty :author ;
5     owl:allValuesFrom :Person ] .
```

[299] suggested an extension of the OWL 2 profile OWL 2 DL to support the formulation of integrity constraints. This enables to use OWL 2 as a constraint language for validating RDF data under the CWA by conjunctive query answering. [286] proposed an alternative semantics for OWL 2 using the CWA so that it could be used to formulate integrity constraints. They examined integrity constraint semantics proposed in the deductive databases literature and adopted them for OWL 2 by reducing the validation based on integrity constraints to SPARQL query answering by means of reasoners. Although the alternative semantics for OWL 2 is implemented in the *Stardog* database,[21] it has never been submitted to an international standards organization such as the W3C.

In DL, reasoning tasks like query answering or detection of inconsistencies require the consideration of knowledge that is not only defined explicitly but also implicitly. To do so there are two different ways called *forward-* and *backward-chaining*. The first method implies a materialized knowledge base, where the original knowledge base is extended by all assertions that can be inferred. State-of-the-art DL or OWL reasoners following this approach are *FaCT++* [306], *Pellet* [285], *RacerPro* [132], and *HermiT* [153].

On the second approach, the original knowledge base is kept in its original state. Before queries are evaluated against the knowledge base, queries are rewritten such that the rewritings also consider the implicit knowledge in the result set. Approaches following this way are *PerfectRef* [75] or *TreeWitness* [189], which are implemented in the –ontop– framework[22] for ontology-based

[21] http://stardog.com
[22] http://ontop.inf.unibz.it

data access. The first solution is applied on local knowledge bases whereas the second is more appropriate for federative environments like in [238, 239].

Stardog Integrity Constraint Validation (ICV) and the *Pellet Integrity Constraint Validator (ICV)* use OWL 2 constructs to formulate constraints on RDF data. *Pellet ICV*[23] is a proof of concept extension for the OWL 2 DL reasoner *Pellet* [285]. *Stardog ICV*[24] validates RDF data stored in a Stardog database according to constraints in SPARQL, OWL 2, or SWRL [154].

2.4.5 Shape Expressions (ShEx)

Shape Expressions (ShEx) [33, 119, 260, 287] specifies a schema language for RDF graphs designed to provide a high-level, user friendly syntax with intuitive semantics similar to the syntax and the semantics of regular expressions. ShEx allows to describe the vocabulary and the structure of an RDF graph, and to constrain the allowed values for properties. It includes an algebraic grouping operator (in a first version of ShEx, the conjunction operator was used instead of grouping [263]), a choice operator, cardinality restrictions on properties, negation, and recursion [34]. [288] define the formal semantics of the ShEx language through the use of the formal specification language *Z notation* [1, 163]. [34] illustrate the features of the language and demonstrate these features with examples.

ShEx enables to validate nodes in a graph against schema constructs called *shapes*, i.e., graphs with labeled patterns which are used to express formal constraints on the content of data graphs. Validating a focus node, i.e., the currently validated node, in a graph against a shape recursively tests the nodes which are the subjects or objects of triples constrained in that shape. Given a node in an RDF graph and a constraint defined in a ShEx schema, the ShEx validation algorithm checks whether the node satisfies that constraint. The algorithm outputs a proof including associations of nodes and the constraints that they satisfy.

[292] studied the complexity of validation with ShEx in absence of negation and under the closed world assumption. The language's balance between expressivity and complexity is supplemented by an extension mechanism which enables more expressive semantic actions using SPARQL, acting like Schematron rules embedded in XSD or Relax NG.

ShEx addresses the problem of detecting and terminating cyclic validation: If a graph has cycles on edges which appear in shapes, validation may arrive at validating the same node against the same shape, e.g., when testing if a publication and all the referenced publications are valid.

ShEx allows to restrict individuals of given classes to have property relationships of all properties of exactly one of multiple mutually exclusive property groups. Publications, e.g., are either identified by an ISBN and a title (for

[23] http://clarkparsia.com/pellet/icv
[24] http://docs.stardog.com/#_validating_constraints

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:

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

In case *Harry-Potter* is a publication with an ISBN and a title without an ISSN, *Harry-Potter* is considered as a valid publication.

ShEx has several open source implementations[25] (some of them can be tested online), has been used as a basis for web applications like the web application developed by [133], has been used for the description and validation of two linked data portals [120], and is currently used in medical informatics for describing clinical models.

As future development of ShEx, [34] are working on the definition of high-level logical constraints on top of ShEx schemas, on data exchange and data transformation solutions based on ShEx, and on heuristics for helpful error reporting.

2.4.6 Resource Shapes (ReSh)

ShEx was originally created to provide a domain-specific language for Resource Shapes, a technique developed by IBM as part of the *Open Services for Lifecycle Collaboration (OSLC)* initiative [173]. *Resource Shapes (ReSh)* [111, 274] defines its own vocabulary for specifying shapes of RDF resources and for describing simple conjunctions of shape constraints. [275] define *shape* as a description of the set of triples a resource is expected to contain and as a description of the integrity constraints those triples are required to satisfy. ReSh includes descriptive features like *oslc:readOnly*, *oslc:hidden*, and *oslc:name* which have no effect on validation but can be useful for tools generating user input forms.

It is a common requirement to narrow down the value space of properties by an exhaustive enumeration of valid values. Consider, e.g., the following constraint stating that books on the topic computer science can only have "Computer Science", "Informatics", and "Information Technology" as subjects:

```
ReSh:
:Computer-Science-Book a oslc:ResourceShape ;
    oslc:property [ oslc:propertyDefinition :subject ;
        oslc:allowedValues [ oslc:allowedValue
            "Computer Science" , "Informatics" , "Information Technology" ] ] .
```

[25] http://www.w3.org/2001/sw/wiki/ShEx#Implementations

2.4.7 Description Set Profiles (DSP)

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

A *Dublin Core Application Profile (DCAP)* [86] defines metadata records which meet specific application needs while providing semantic interoperability with other applications on the basis of globally defined vocabularies and models. The *Singapore Framework for Dublin Core Application Profiles* [235] is a framework for designing metadata and for defining DCAPs. The framework comprises descriptive components that are necessary or useful for documenting DCAPs.

The *DCMI Abstract Model* [258] is required to formalize a notion of machine-processable application profiles. It specifies an abstract model for Dublin Core metadata which is independent of any particular encoding syntax. Its primary purpose is to specify the components used in Dublin Core metadata. The *DC-RDF* [236] recommendation depicts how Dublin Core metadata and therefore the constructs of the DCMI Abstract Model are represented in RDF by means of the abstract syntax of the RDF model.

The *Description Set Profile (DSP)* [234] is a generic constraint language which is used to formally specify structural constraints on sets of resource descriptions within an application profile. DSP allows to restrict resources that may be described by descriptions in a description set, the properties that may be used, and the values the properties may point to. DSP constraints can be represented by means of an RDF vocabulary or a conventional XSD.

With DSP, constraints on resources can be defined within an application profile, i.e., constraints expressed in DSP determine how valid descriptions of resources in a description set should look like. A DSP consists of a set of *dsp:DescriptionTemplate*s that put constraints on classes denoted by *dsp:resourceClass*:

```
DSP:
[ a dsp:DescriptionTemplate ;
  dsp:resourceClass :Science-Fiction-Book ;
  dsp:statementTemplate [
      a dsp:NonLiteralStatementTemplate ;
      dsp:property :subject ;
      dsp:nonLiteralConstraint [
          a dsp:NonLiteralConstraint ;
          dsp:valueClass skos:Concept ;
          dsp:valueURIOccurrence "mandatory"^^dsp:occurrence ;
          dsp:valueURI :Science-Fiction, :Sci-Fi, :SF ;
          dsp:vocabularyEncodingSchemeOccurrence "mandatory"^^dsp:occurrence ;
          dsp:vocabularyEncodingScheme :Science-Fiction-Book-Subjects ; ] ] .
```

The description template above describes resources of the type *Science-Fiction-Book* (*dsp:recourceClass*). The DSP construct *dsp:NonLiteralStatement Template* is used to specify constraints on object properties with a particular resource class as domain. The *dsp:NonLiteralStatementTemplate* in the example restricts science fiction books to have *subject* (*dsp:property*) relationships to non-literal values which are further restricted in a *dsp:NonLiteralConstraint*.

Non-literal values, to which the property *subject* is pointing, have to be of the class *skos:Concept* (*dsp:valueClass*). A URI must be given (*dsp:valueURI Occurrence* mandatory) for non-literal values, whereas the only allowed URIs (*dsp:valueURI*) are *Science-Fiction*, *Sci-Fi*, and *SF*.

Controlled vocabularies like *Science-Fiction-Book-Subjects* are represented as *skos:ConceptSchemes* in RDF and as *dsp:VocabularyEncodingSchemes* in DSP. If vocabulary encoding schemes must be stated (*dsp:vocabularyEncoding SchemeOccurrence* mandatory), they must contain the non-literal values specified within the description template.

In this case, non-literal values must be assigned to the class *skos:Concept* and be related to the controlled vocabulary *Science-Fiction-Book-Subjects* via the object properties *skos:inScheme* and *dcam:memberOf*. The book *The-Time-Machine* satisfies all constraints defined for resources of the type *Science-Fiction-Book*:

```
RDF (Turtle syntax):
:The-Time-Machine
    a :Science-Fiction-Book ;
    :subject :Science-Fiction .
:Science-Fiction
    a skos:Concept ;
    dcam:memberOf :Science-Fiction-Book-Subjects ;
    skos:inScheme :Science-Fiction-Book-Subjects .
:Science-Fiction-Book-Subjects
    a skos:ConceptScheme .
```

2.4.8 BIBFRAME

BIBFRAME [125, 195, 220, 223] has been developed by the *Bibliographic Framework Initiative*, a framework or lightweight meta-model for the discovery and exchange of library information. BIBFRAME defines a vocabulary [205, 207] having a strong overlap with DSP and *BIBFRAME Profiles* [206] are essentially identical to DCAPs. According to the BIBFRAME Profile below, persons must have a name which has to be a literal:

```
BIBFRAME:
{ "Profile":
  {
    "id": "profile:bf:Persons",
    "title": "BIBFRAME Persons",
    "resourceTemplates": [
      {
        "id": "profile:bf:Person",
        "resourceURI": :Person,
        "resourceLabel": "Person",
```

```
11          "propertyTemplates": [
12            {
13              "propertyURI": :name,
14              "propertyLabel": "name",
15              "mandatory": "true",
16              "type": "literal" } ] } ] } }
```

A BIBFRAME Profile is a document, or set of documents, that puts a profile, e.g., for local cataloging practices, into a broader context of functional requirements, domain models, guidelines on syntax and usage, and possible data formats. This BIBFRAME Profile document describes an information model and reference serialization to support a means for identifying and describing structural constraints. A profile defines a means in which a resource can be constrained by enumerating the properties that may be used to describe it and by defining which property values may be given.

A BIBFRAME Profile is primarily a means for an application, e.g., a cataloging tool, to guide a cataloger in the creation or modification of a BIBFRAME record. As a BIBFRAME Profile contains formal syntactic constraints only, it is needed to be combined with human-readable information, semantic expressions, and usage guidelines in order to be fully useful for a community [206].

2.4.9 Schemarama

Schemarama [102] is a validation technique for specifying the types of subgraphs connected to a particular set of nodes in an RDF graph. Schemarama allows to check if nodes in an RDF graph have certain required properties. Schemarama is based on Schematron and the *Squish RDF Query language* [222], an SQL-like query language for RDF, instead of SPARQL.

Schemarama is very similar to Schematron. It has two main parts, which are represented as Squish queries. The first is the context Squish query used to identify the nodes to be validated. The second is the test Squish query that performs tests on the selected nodes. Error messages are displayed in case selected nodes are not successfully validated against these tests [222].

2.4.10 Shapes Constraint Language (SHACL)

The W3C RDF Data Shapes Working Group currently develops *SHACL* [35, 186, 187, 261], the *Shapes Constraint Language*,[26] a language for describing and constraining the contents of RDF graphs. SHACL provides a high-level RDF vocabulary similar to but more expressive than ReSh for describing shape constraints. SHACL can be used (1) to communicate information about data structures associated with some process or interface, (2) to generate data,

[26] Open source reference implementation online available at: https://github.com/TopQuadrant/shacl

(3) to validate data, and (4) to drive user interfaces. [186] define the SHACL language and its underlying semantics.

SHACL groups descriptive information and constraints that apply to a given data node into *shapes*. SHACL defines what it means for an RDF graph containing the data that is to be validated, referred to as the *data graph*, to conform to the *shapes graph*, which includes shape definitions.

SHACL can be used to determine whether data graphs, containing information about books and their authors, conform to the following constraints: (1) Each book is either published or not (yet) published. (2) Books must have at least one author. (3) Each author must be a person. (4) Persons must have precisely one name which must be of the datatype string. A shapes graph that defines these constraints has two shapes. The first, *BookShape*, contains the three constraints on books. The second, *PersonShape*, encompasses the constraint on persons:

```
SHACL Shapes Graph:
:BookShape
    a sh:Shape ;
    sh:scopeClass :Book ;
    sh:property [
        sh:predicate :isPublished ;
        sh:allowedValues (true false) ;
        sh:minCount 1 ;
        sh:maxCount 1 ; ] ;
    sh:property [
        sh:predicate :author ;
        sh:valueShape :PersonShape ;
        sh:minCount 1 ; ] .
:PersonShape
    a sh:Shape ;
    sh:scopeClass :Person ;
    sh:property [
        sh:predicate :name ;
        sh:datatype :string ;
        sh:minCount 1 ;
        sh:maxCount 1 ; ] .
```

A shape may include a *scope* defining which data nodes must conform to the shape. The two example shapes include scope information which in this case says that their constraints apply to all nodes that have an *rdf:type* link to the classes *Book* and *Person* respectively. The property *rdf:type* is used to determine which shapes a given node is expected to fulfill. The scope includes all instances of the *sh:scopeClass* and its sub-classes, by following *rdfs:subClassOf* links. The following data graph might be validated against this shapes graph:

```
Data Graph:
:The-Adventures-Of-Sherlock-Holmes
    a :Book ;
    :isPublished true ;
    :author :Doyle .
:Doyle
    a :Person ;
    :name "Doyle" .
:The-Hobbit
```

```
10      a :Book ;
11      :isPublished "yes" ;
12      :author :Tolkien .
13  :Tolkien
14      a :Person ;
15      :name "Tolkien", "J.R.R. Tolkien" .
```

Conformance to the shapes graph can be programmatically checked by processors, i.e., by *SHACL engines*. The process of checking conformance is referred to as *validation*. When a data node is checked for conformance to a shape, that node is called the *focus node*. The SHACL engine validates the nodes *The-Adventures-Of-Sherlock-Holmes* and *The-Hobbit* against the shape *BookShape*. Validating the first node determines that *The-Adventures-Of-Sherlock-Holmes* satisfies the constraints in *BookShape*, along the way determining that its author *Doyle* satisfies the constraints in *PersonShape*. Validating the third node determines that *The-Hobbit* violates the constraint on values for the predicate *isPublished*, since *"yes"* is not contained in the list of the allowed values. *The-Hobbit* also violates the constraint on values for the predicate *author*, since *Tolkien* violates the *PersonShape* constraint on the maximum number of values for the predicate *name*.

The output of a SHACL constraint validation process is a set of validation results, a *validation report* which indicates whether or not the data graph conforms to the shapes graph. If any constraints are not satisfied, then the validation report includes one or more *violations* indicating the source of the problem. SHACL includes an RDF vocabulary to represent such validation results together with structural information that may provide guidance on how to fix a violation, as well as human-readable messages.

In addition to the high-level vocabulary SHACL provides, *native constraints* can be associated with shapes using SPARQL and similar execution languages like JavaScript. Native constraints in a language like SPARQL typically provide a lot of flexibility. However, SPARQL-based constraints may be hard to understand and repetitive.

Templates can be used to encapsulate and parameterize such native queries. Templates make it possible to generalize, so that constants get substituted by arguments. This allows the query logic to be reused in multiple places, without having to write any new SPARQL query. SHACL *functions* can be called within SPARQL queries to encapsulate complex logic of other SPARQL queries or executable logic in other languages.

Various W3C standards, including RDFS and OWL, provide semantic interpretations for RDF graphs that allow additional RDF statements to be inferred from explicitly given assertions. What the correct answers to a SPARQL query are, depends on the used *entailment regime* [124]. SPARQL 1.1 [134] defines a set of conditions that have to be met when defining what correct results are for SPARQL queries under different entailment regimes. By default, SHACL does not assume any entailment regime to be activated on the data

graph. However, a SHACL engine can be instructed to ensure that a given entailment is activated.

2.5 RDF Validation Semantics

In this section, we (1) define what semantics is and provide intuitive examples for each semantic assumption, (2) compare the semantics for RDF and XML validation on the one side and for OWL reasoning on the other side, and (3) explain why OWL can be considered as a constraint language under the same semantics adopted when validating RDF data.

A *knowledge base* is a set of *sentences*. These sentences are related but not identical to the sentences of English and other natural languages. Each sentence is expressed in a knowledge representation language and according to its *syntax*. The *semantics* or meaning of sentences defines the truth of each sentence which must be either true or false [273].

The validation of RDF and XML data and reasoning in OWL assume different semantics (see Figure 2.1) 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*).

Table 2.1. Semantics for RDF Validation, XML Validation, and OWL Reasoning

	RDF Validation	XML Validation	OWL Reasoning
CWA vs. OWL	CWA	CWA	OWA
UNA vs. nUNA	UNA	UNA	nUNA

2.5.1 Closed-World Assumption vs. Open-World Assumption

The ambiguity in semantics is one of the main reasons why OWL has not been adopted as the standard constraint language for RDF validation in the past. Reasoning in OWL is based on the *open-world assumption* (*OWA*) [224, 267], i.e., a statement cannot be inferred to be false if it cannot be proved to be true which fits its primary design purpose to represent knowledge on the World Wide Web. As each book must have a title and *The-Odyssey* is a book, there must be a title for *The-Odyssey* as well. In an OWA setting, this constraint does not cause a violation, even if there is no explicitly defined title, since there must be a title for this book which we may not know.

As RDF validation has its origin in the XML world, many RDF validation scenarios require the *closed-world assumption* (*CWA*) [74, 224, 267], i.e., a statement is inferred to be false if it cannot be proved to be true. Thus, classical constraint languages are based on the CWA where constraints need to be satisfied only by named individuals. In the example, the CWA yields to a violation in case there is no explicitly defined title for the book *The-Odyssey*.

Publications must have at least one author is another example of a constraint for which changed underlying semantics leads to differences regarding validation results. In a CWA setting, a publication without an explicitly stated author violates the constraint, whereas with OWA semantics, a publication without an explicitly stated author does not raise a violation as the constraint entails that there must be an author for a particular publication which we may not know.

2.5.2 Unique Name Assumption vs. Non-Unique Name Assumption

OWL is based on the *non-unique name assumption* (*nUNA*) whereas RDF validation requires that different names represent different objects (*unique name assumption* (*UNA*)) [300]. Although OWL does not assume the UNA, OWL has the constructs *owl:sameAs* and *owl:differentFrom* to explicitly state that two names are the same or different. For *functional properties*, e.g., it makes a difference with regard to validation results if the UNA or the nUNA is assumed. As the property *title* is functional, a book can have at most one distinct title. UNA causes a clash if the book *One-Thousand-And-One-Nights* has more than one title. For nUNA, however, reasoning concludes that the title *One-Thousand-And-One-Nights* must be the same as the title *Tausendundeine-Nacht* which resolves the violation.

Assuming the OWA and the nUNA when validating RDF data would limit validation possibilities. RDF validation won't be that restrictive and therefore we won't get the intended validation results. This is the reason why (1) [82, 228, 300] propose the use of OWL expressions with the CWA to express integrity constraints, (2) [300] suggest an alternative semantics for OWL using the CWA and the UNA so that it could be used to validate integrity constraints, and (3) [254] claims that Description Logics and therefore OWL axioms can be interpreted in a closed-world setting and used for constraint checking.

When using OWL axioms in terms of constraints, we adopt the same semantics that is used when validating RDF data. The main criticism against such an approach is that it associates an alternative semantics with the existing OWL syntax, which can be misleading for users. Note that in any case, OWL inference engines cannot be used for checking constraints and that a dedicated implementation is required.

3

Vocabularies for Representing Research Data and its Metadata

The *social, behavioral, and economic (SBE) sciences* require high-quality data for their empirical research. For more than a decade, members of the SBE sciences community have been developing and using a metadata standard, composed of almost twelve hundred metadata fields, known as the *Data Documentation Initiative (DDI)* [95], an XML format to disseminate, manage, and reuse data collected and archived for research [316]. In XML, the definition of schemas containing constraints on data and the validation of data according to these constraints is commonly used to ensure a certain level of data quality. With the rise of the Web of Data, data professionals and institutions are very interested in having their data be discovered and used by publishing their data directly in RDF or at least accurate metadata about their data to facilitate data integration.

There are different types of research data and related metadata, but because of the lack of respective RDF vocabularies, just a few of them can be expressed in RDF. Having these missing vocabularies enables (1) to represent all types of research data and its metadata in RDF and (2) to validate RDF data according to constraints extractable from these vocabularies.

To close this gap by developing suitable RDF vocabularies, the *RDF Vocabularies Working Group*,[1] an international working group hosted by the international standards organization *DDI Alliance*,[2] has been established in 2011. Being part of the working group and in collaboration with Linked Data community members and statistical domain experts, i.e., representatives of national statistical institutes and national data archives as well as core members of the DDI Alliance Technical Committee,[3] we have developed three vocabularies:

- The *DDI-RDF Discovery Vocabulary (DDI-RDF)* [42] is based on the XML standard DDI and supports the discovery of metadata on *unit-record*

[1] http://www.ddialliance.org/alliance/working-groups#RDF
[2] http://www.ddialliance.org
[3] http://www.ddialliance.org/alliance/working-groups#tc

data, the type of data most often used in research within the SBE sciences, i.e., data collected about individuals, businesses, and households. It can be applied to research data from many different domains, rather than being specific to a single set of domain data.

- *Physical Data Description (PHDD)* [321] is a vocabulary to describe data in tabular format and its physical properties. The data could either be represented in form of records with character-separated values (CSV) or with fixed length.
- The *SKOS Extension for Statistics (XKOS)* [84] is a vocabulary to describe the structure and textual properties of formal statistical classifications as well as relations between classifications and concepts, and to introduce refinements of SKOS semantic properties to allow the use of more specific relations between concepts.

Throughout this thesis, several RDF vocabularies for representing different types of research data and related metadata (DDI-RDF, SKOS, QB) as well as various DDI XML specifications (DDI-Codebook, DDI-Lifecycle, DDI-Lifecycle MD) serve as intuitive case studies for (1) validating RDF data and (2) the in Section 4 proposed approach enabling to directly validate XML using common RDF validation tools.

In the remainder of the chapter, we first give an overview of the types of research data and related metadata and which vocabularies are commonly used to represent them in RDF (see Section 3.1). Second, we define each type of research data and related metadata as stated by data professionals, delineate their transition in the research data lifecycle, and depict how metadata on each type of research data is represented using XML standards before respective RDF vocabularies have been developed. The focus lies on unit-record data, since with DDI-RDF we have developed a vocabulary to represent metadata on this kind of research data. We conclude this section by presenting the model-driven further development of the DDI standard itself whose requirements are oriented on experiences made with DDI-RDF (see Section 3.2).

Section 3.3 serves to provide a detailed depiction of how metadata about unit-record data is represented using DDI-RDF. First, we separately look at the motivations having such a vocabulary at hand for individuals from (1) the Linked Data community and (2) data archives, research institutes, data libraries, and government statisticians. Second, we present an overall picture of the conceptual model of the vocabulary. When defining the conceptual model, we reuse existing vocabularies wherever possible and extend them where needs are almost but not completely covered. We base the design of the conceptual model on use cases that represent real information needs of researchers. Since DDI-RDF only covers a small subset of the underlying XML standard DDI for the purpose of discovery, it is worthwhile to have relationships to and from original DDI-XML files.

3.1 Common Vocabularies

In this chapter, we give an overview of the types of research data and related metadata and present in form of a complete representative running example[4] which vocabularies[5] are commonly used to represent each type in RDF.

3.1.1 Unit-Record Data and Aggregated Data

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

By its nature, unit-record data is highly confidential and access is often only permitted for qualified researchers who must apply for access. Researchers typically represent their results as aggregated data in form of multi-dimensional tables with only a few columns, so-called *variables* such as *sex* or *age*. Aggregated data, which answers particular research questions, is derived from unit-record data by statistics on groups or aggregates such as counts, frequencies, and arithmetic means. Aggregated data is often generated by tabulating or aggregating unit-record data sets. For example, if an observation in a census table indicates the population of a certain age group in a particular region, then this fact was obtained by aggregating that number of individual records from a unit-record data set. The purpose of publicly available aggregated data is to get a first overview and to gain an interest in further analyses on the underlying unit-record data. Aggregated data is published in form of CSV files, allowing to perform further calculations on the data.

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

[4] Complete running example in RDF on GitHub
[5] RDF vocabularies commonly used to represent different types of research data and related metadata on GitHub

age, e.g., may be represented by values of the datatype *xsd:nonNegativeInteger* or a code list of age clusters (e.g., '0 to 10' and '11 to 20').

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

3.1.2 Vocabulary to Represent Aggregated Data and its Metadata

The *RDF Data Cube Vocabulary (QB)* [88] is a W3C recommendation for representing *data cubes*, i.e., multi-dimensional aggregated data and its metadata, in RDF [87]. A *qb:DataStructureDefinition* contains metadata on the data collection. The variable *formal childcare* is modeled as *qb:measure*, since it stands for what has been measured in the data collection. *Year*, *duration*, *age*, and *country* are *qb:dimensions*. The actual data values, i.e., the availability of childcare services in percent over the population, are collected in a *qb:DataSet* representing an aggregated data set. Each data value is represented inside a *qb:Observation* containing the values for each dimension [91].

A well-formed *RDF Data Cube* is an RDF graph describing one or more instances of *qb:DataSet* for which each of the 22 integrity constraints, defined within the QB specification, passes [88]. Each integrity constraint is expressed as narrative prose and, where possible, as a SPARQL ASK query or query template. If the ASK query is applied to an RDF graph then it will return true if that graph contains one or more QB instances which violate the corresponding constraint.

The development of QB is based on *Statistical Data and Metadata eXchange (SDMX)* [161, 290], an XML standard to describe aggregated data, and on the *Statistical Core Vocabulary (SCOVO)* [87, 141, 142], an RDFS-based, lightweight, and extensible vocabulary for representing statistical data on the Web. SCOVO offers a basic core of classes and properties for representing data sets, multiple dimensions, and statistical items. [319] extend SCOVO with a vocabulary enabling the connection of SCOVO-described data sets with external vocabularies to perform more complex data analyses and improve discoverability and reusability.

3.1.3 Vocabulary to Represent Metadata on Tabular Data

QB provides for the description of the structure of data cubes, but also for the representation of the cube data itself, that is, the observations that make up the cube data set [87]. This is not the case for DDI-RDF, which only describes the structure of a data set, but is not concerned with representing the actual data in it, which is assumed to sit in a data file, e.g., in a CSV file or in a proprietary statistical package file format. To describe such data, we have developed the DDI based *Physical Data Description (PHDD)* [321], a vocabulary to represent metadata about data in tabular format as well as its physical properties in RDF (see Figure 3.1). The data could either be represented in records with character-separated values (CSV) or with fixed length, which allows to perform further calculations on the data.

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

In December 2015, the *CSV on the Web Working Group*[6] has published a group of W3C specifications [296–298], including (1) *CSV on the Web (CSVW)* [301–303], a metadata vocabulary for tabular data, (2) standard methods to find data and metadata, and (3) standard mapping mechanisms transforming CSV to other formats. CSVW comprises rich possibilities for defining constraints on cell values. Even though PHDD has been developed earlier and independently from CSVW, PHDD and CSVW have an overlap in the main description of tabular data in CSV format. Compared to CSVW, PHDD can also be used to describe tabular data with fixed record length, data with multiple records per case, and programs which generated tabular data.

3.1.4 Vocabulary to Represent Metadata on Unit-Record Data

For more detailed analyses, we refer to the unit-record data collected for the series *EU-SILC (European Union Statistics on Income and Living Conditions)*.[7] Where data collection is cyclic, data sets may be released as *series*, where each cycle produces one or more data sets. The aggregated variable *formal childcare* is calculated on the basis of six unit-record variables (i.a., *Education at pre-school*) for which detailed metadata is given (i.a., code lists)

[6] https://www.w3.org/2013/05/lcsv-charter
[7] http://www.gesis.org/en/missy/metadata/EU-SILC

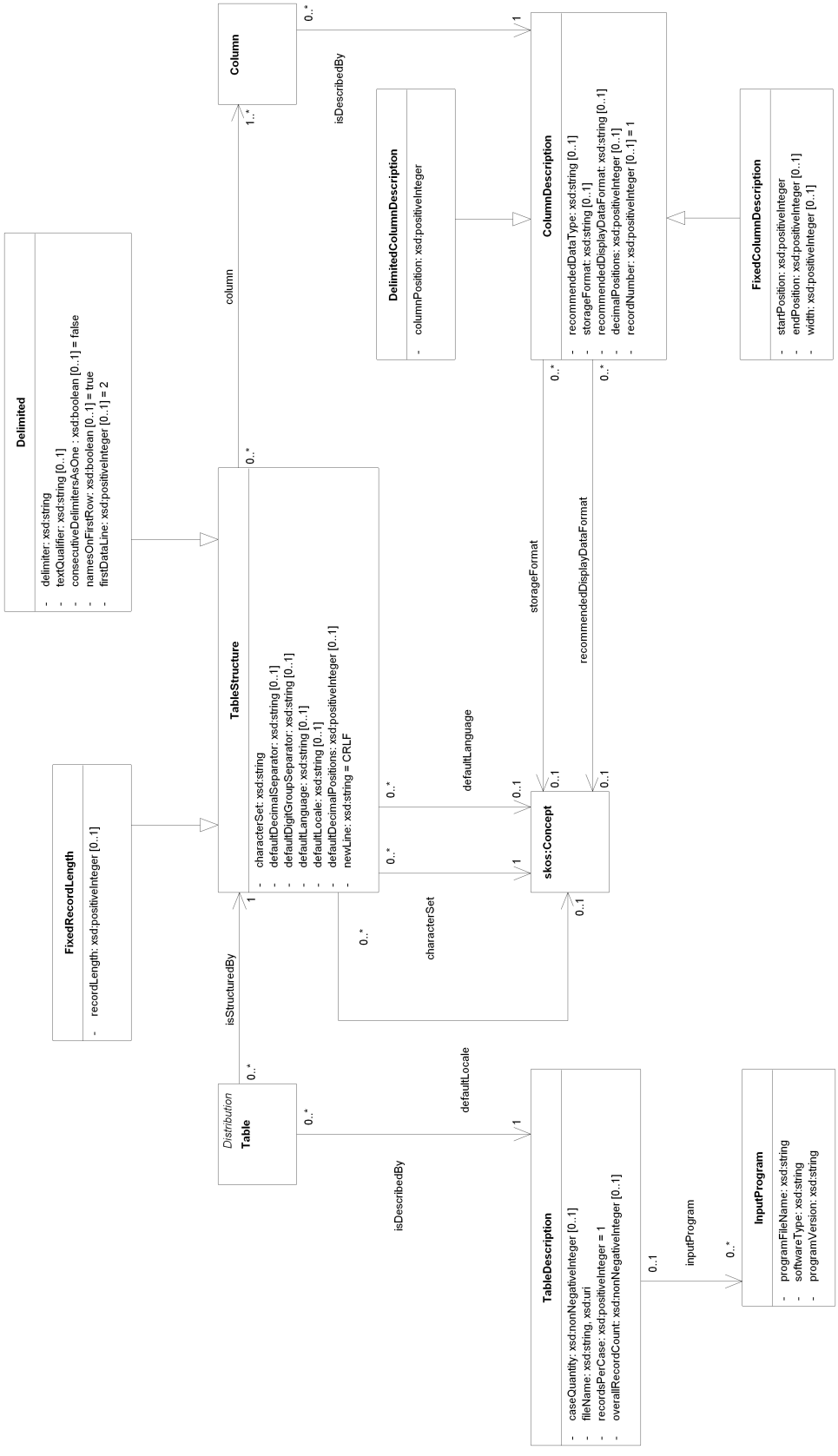

Fig. 3.1. Physical Data Description (PHDD)

enabling researchers to replicate the results shown in aggregated data tables. The *DDI-RDF Discovery Vocabulary (DDI-RDF)* [42] is used to represent metadata on unit-record data in RDF. The series (*disco:StudyGroup*) EU-SILC contains one study (*disco:Study*) for each year (*dcterms:temporal*) of data collection. The property *dcterms:spatial* points to the countries for which the data has been collected. The study *EU-SILC 2011* contains eight unit-record data sets (*disco:LogicalDataSet*) including unit-record variables (*disco:Variable*) like the six ones needed to calculate the aggregated variable *formal childcare*.

3.1.5 Vocabulary to Represent Knowledge Organization Systems

The *Simple Knowledge Organization System (SKOS)* [159, 219] is a vocabulary to represent knowledge organization systems such as thesauri, classification schemes, and taxonomies. Using SKOS, a knowledge organization system is expressible as machine-readable data in a machine-processable standard format for the Semantic Web. It can then be exchanged between computer applications and published in a machine-readable format in the Web [159]. SKOS provides high interoperability with other standards, formats, and applications as it is based on RDF, the standard model for data interchange on the Web. SKOS is used to represent term relations, term hierarchies, and the structure and semantics of vocabularies. A vocabulary is typically represented as a *skos:ConceptScheme* that holds multiple *skos:Concepts* which can be linked to other *skos:Concepts* by hierarchical and associative properties that are oriented on relations of the ISO norms for thesauri like *skos:broader*, *skos:narrower*, and *skos:related*.

Thesauri organize complex relations between terms even on the lexical level. The *SKOS Simple Knowledge Organization System eXtension for Labels (SKOS-XL)* [218] offers additional support for describing and linking lexical entities. This provides a complexity of relationships between terms which is needed by several vocabularies.

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

3.1.6 Vocabulary to Represent Metadata on Formal Statistical Classifications

A formal statistical classification like the *International Standard Classification of Occupations* is a hierarchical concept scheme including concepts, associated numeric codes, short textual labels, definitions, and longer descriptions that include rules for their application. The use of statistical classifications is very common in research data sets - they are treated as SKOS concept schemes in DDI-RDF, but in some cases those working with statistical classifications desire more expressive capabilities than SKOS provides. To support such researchers, we developed *XKOS*, the *SKOS Extension for Statistics* [84] to allow for a more complete description of such classifications [123] and to introduce refinements of SKOS semantic properties [85].

XKOS extends SKOS for the needs of statistical classifications. It does so in two main directions. First, it defines a number of terms that enable to represent statistical classifications with their structure and textual properties as well as different types of relations between classifications and concepts. Second, it refines SKOS semantic properties to allow the use of more specific relations between concepts. These semantic properties can be used for the representation of classifications or any other case where SKOS is employed [123]. While the use of XKOS is not required, DDI-RDF and XKOS are designed to work in complementary fashion, whereby SKOS properties may be substituted by additional XKOS properties.

A question, typically asked by researchers, could be to query all the data sets (*disco:LogicalDataSet*) which have a specific classification and to query on semantic relationships of classifications using XKOS properties. By means of these properties not only hierarchical relations can be queried but also part of relationships (*xkos:hasPart*), more general (*xkos:generalizes*) and more specific (*xkos:specializes*) concepts, and positions of concepts in lists (*xkos:previous*, *xkos:next*).

Figure 3.2 visualizes how XKOS is used to represent statistical classifications in RDF - inspired by *ANZSIC*, the *Australian and New Zealand Standard Industrial Classification*, a classification covering the field of economic activities. A small excerpt is shown here, limited to the classification object itself, its levels, one item of the most detailed level (<*L672000*>), and its parent items. Note that the URIs employed in this example are entirely fictitious, since the ANZSIC has not yet been published in RDF.

On the left of the figure is the skos:ConceptScheme instance that corresponds to the *ANZIC 2006* classification scheme, with its various SKOS and DC properties. Additional XKOS properties indicate that the classification has four levels (*xkos:ClassificationLevel*) and covers (*xkos:covers*) the field of economic activities, represented here as a concept from the *EuroVoc* thesaurus. In this case, the coverage is intended to be exhaustive and without overlap, so *xkos:coversExhaustively* and *xkos:coversMutuallyExclusively* could have been used together instead of *xkos:covers*.

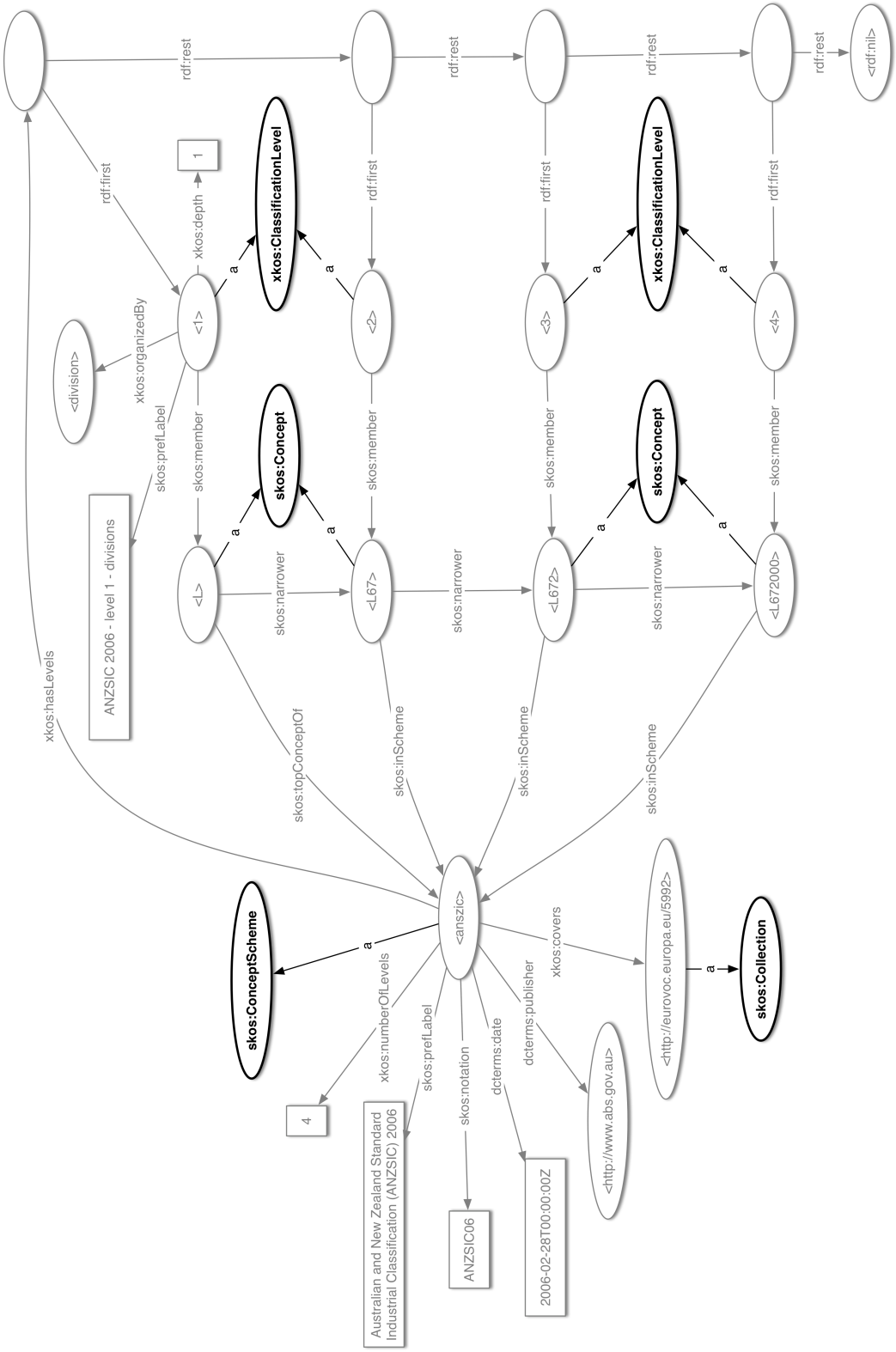

Fig. 3.2. Representing the Formal Statistical Classification ANZSIC in RDF

The four levels are organized as an rdf:List which is attached to the classification by the *xkos:levels* property. Some level information has been represented on the top level, for example its depth in the classification (*xkos:depth*) and the concept that characterizes the items it is composed of (*xkos:organizedBy*). In the same fashion, concepts of subdivision, group, and class could be created to describe the items of the lower levels.

The usual SKOS properties connect the classification items to their respective level (*skos:member*) and to the classification (*skos:inScheme* or its specialization *skos:topConceptOf*) for the items of the first level. Similarly, *skos:narrower* expresses the hierarchical relations between the items, but the sub-properties defined in the XKOS specification could also be used instead. For example, *xkos:hasPart* could express the partitive relation between subdivisions and groups. For clarity, the properties of the classification items (code, labels, notes) have not been included in the figure.

3.1.7 Vocabulary to Represent Metadata on Data Sets Inside of Data Collections

The *Data Catalog Vocabulary (DCAT)* [212] enables to describe data sets inside of data collections like portals, repositories, catalogs, and archives which serve as typical entry points when searching for data. DCAT makes few assumptions about the kind of data sets being described and focuses on general metadata about data sets and different ways of distributing and accessing data sets, including their availability in multiple formats.

Users search for aggregated and unit-record data records (*dcat:Catalog Record*) in data catalogs (*dcat:Catalog*). As search differs depending on users' information needs, they may only search for records' metadata (e.g., *dcterms:title*, *dcterms:description*), or they may formulate more sophisticated queries on aggregated and unit-record data sets (*dcat:Dataset*) or their distributions (*dcat:Distribution*) which are part of the records. Users often search for data sets covering particular topics (*dcat:keyword*, *dcat:theme*), time periods (*dcterms:temporal*), locations (*dcterms:spatial*), or for certain formats in which the data distribution is available (*dcterms:format*).

The combined usage of PHDD, DDI-RDF, and DCAT enables the creation of data repositories providing metadata for the description of collections, data discovery, and the processing of the data. Descriptions in PHDD could be added to web pages which enables an automatic processing of the data by programs.

3.2 Representing Metadata on Research Data

For data professionals - researchers, data librarians, and statisticians - the term *data* refers to some very specific types of what most people think of as data. For those in the Linked Data community, data is a very broad term indeed, embracing basically anything accessible on the Web. In developing an

RDF vocabulary for describing research data, it is important to understand the narrower "professional" definition of data, since it is this which is to be described using the new vocabulary, not data in the more general sense. The narrower term, as used by data specialists, is what is meant unless otherwise stated.

So, what does the term *data* mean? It actually has several distinct meanings: raw data; unit-record data (also called microdata); and aggregated data (also called tabulated data). We characterize each type here, as they serve different purposes. In order to understand them, it is important to understand how each fits into the data lifecycle, as data is collected and processed to support research.

Raw data[8] refers to the set of numbers and other values (often coded values coming from concept schemes or classifications) which are the direct input into the research process. These are often the result of surveys. For instance, each person responding to a survey might be asked: "What is your gender?". The answer would be either *Male* or *Female*, which is a very simple example of a concept scheme for gender. In the data set, the responses might be recorded as *1* (for Male) or *2* (for Female). Raw data can also come from other sources, such as devices performing measurements, or from administrative registers, i.e., databases containing information collected for non-research purposes, but which are useful for research, such as registers of births and deaths, and clinical systems in hospitals.

Once collected, raw data is processed further, to clean up values which are wrong or likely to distort the research. Some other values are processed into a form which is easy to work with. If there are missing values, these need to be handled: for instance, it has to be determined why they are missing. The result of this processing is a useful *unit-record data* set.

The structure of unit-record data is very specific: think of a table, where each column contains a particular type of value (e.g., gender, age, or a response to a particular question in a survey) and each row represents the responses for a single unit (typically an individual, a household, or a business). By further processing the (usually large) number of cases, a different type of data is produced – *aggregated* or *tabulated data*.

Take, for example, the following unit-record data set, recording gender, age, highest education degree attained, and current place of habitation:

Table 3.1. Unit-Record Data Set

Case ID	Gender	Age	Degree	Habitation
1	Male	22	High School	Arizona
2	Female	36	PhD	Wisconsin
3	Male	50	PhD	New Mexico
4	Female	65	PhD	Texas
5	Male	23	BA	Vermont

[8] http://stats.oecd.org/glossary

In analyzing this data set, we might decide that older people tend to have higher degrees of education, since no one under the age of 36 has a PhD, and of those under the age of 36, only 50% have a college degree of any type. We could aggregate (or tabulate) the unit-record data into the following aggregated data set:

Table 3.2. Aggregated Data Set

Age	% with High School	% with BA	% with PhD
Age < 36 years	50	50	0
Age >= 36 years	0	0	100

Although we have too few cases, we can see that by focusing on some of the columns in our unit-record data, we can create a table: in this case, age by educational degree. Such tabulations are used by researchers analyzing data to prove or disprove research hypotheses. Tabulations are also created by government statisticians working to support policy decisions.

When we consider the types of data which exist on the Web, and which could be represented on the Web as the result of open data initiatives, we can see that at least the second categories (unit-record data and aggregated data) would be very useful, and in some cases even raw data might be useful, as in the case of government expenditures, for example.

It is very important to understand the distinctions between these types of data, because they are useful for different purposes. Aggregated data can be used to draw many useful types of charts and visualizations, but this cannot be done usefully with unit-record data or raw data, to make a simple point. For most of us, the aggregated data is very helpful and easy to understand – unit-record data requires a higher degree of statistical literacy to make sense of. Both, unit-record and aggregated data is understood as *research data*. This means any data which is used for research not just data which is collected for research purposes.

When working with data of any type, it is not the data alone which is needed – also required to understand and analyze data is a very detailed level of metadata. *Metadata* is structured information that describes, explains, locates, or otherwise makes it easier to retrieve, use, or manage data [232]. How is a column in my unit-record data defined? Where do the values come from? What was the population being surveyed, and how was it selected? This type of metadata includes, but goes well beyond the metadata found in Dublin Core, for example. It is highly specialized, and is specific to the type of data being described.

Within the world of data professionals, two metadata standards have emerged which are becoming widely adopted. For raw data and unit-record data, the metadata standard is called the *Data Documentation Initiative (DDI)* (see Section 3.2). For metadata on aggregated data, the XML standard is known as the *Statistical Data and Metadata eXchange (SDMX)* [161, 290]. [130] describe these two metadata standards, their scope, overlap, and differ-

ences. They also provide some information about which of them may be more appropriate for use in a particular application. In addition, [307] explores their relationship and interoperability. SDMX focuses on processing and exchanging aggregated data, DDI on documenting the aggregation processes, in case they are of interest to researchers. Guidelines recommend practices for creating interoperable data sets and related metadata using SDMX which enables a more efficient exchange of comparable data and metadata [291].

The RDF Data Cube Vocabulary (QB) is based on SDMX. QB maps the SDMX information model to an ontology and is therefore compatible with the cube model that underlies SDMX. DDI and SDMX have traditionally made efforts to align their content [130]. Similarly, some of the developers of the DDI-RDF vocabulary were also involved in the development of QB, allowing the RDF versions of these standards to retain that alignment.

3.2.1 Data Documentation Initiative

The *Data Documentation Initiative (DDI)* [95] is an acknowledged international standard for the documentation and management of unit-record research data in the SBE sciences. DDI describes social sciences data, data covering human activity, and other data based on observational methods measuring real-life phenomena [316, 320].

Archives and data libraries have no control over the form of the data deposited with them by researchers, which is reflected by the DDI standard – it is an XML format for the large amount of metadata needed to understand the wide range of data formats used by researchers at a very detailed level. DDI metadata accompanies and enables data conceptualization, collection, processing, distribution, discovery, analysis, re-purposing, and archiving. DDI does not invent a new model for statistical data. It rather formalizes state-of-the-art concepts and common practice in this domain.

Where a metadata standard such as Dublin Core has dozens of metadata fields, DDI has almost twelve hundred. The metadata is sufficient to support a wide range of uses, including management of data holdings within archives, discovery and dissemination, transformation of the data between different proprietary software formats, and a thorough documentation of the data and how and why it was collected. The key to the reuse and management of data is always metadata, and this has been a major theme within the SBE community for many years.

DDI focuses on both unit-record and aggregated data, but it has its strength in unit-record data. Aggregated data (e.g., multidimensional tables) are likewise covered by DDI. DDI provides summarized versions of the unit-record data in the form of statistics like means or frequencies. Publicly accessible metadata of good quality is important for finding the right data. This is especially the case when access to unit-record data is restricted as a disclosure risk of the observed people exists.

Goals

DDI supports technological and semantic interoperability in enabling and promoting international and interdisciplinary access to and use of research data. Structured metadata with high quality enables secondary analyses without the need to contact the primary researcher who collected the data. Comprehensive metadata (potentially along the whole data lifecycle) is crucial for the replication of analysis results in order to enhance the transparency. DDI enables the reuse of metadata of existing studies (e.g., questions, variables) for designing new studies, an important ability for repeated surveys and for comparison purposes [25, 221].

Users

A large community of data professionals, including data producers (e.g., of large academic international surveys), data archivists, data managers in national statistical agencies and other official data producing agencies, and international organizations use DDI.

It should be noted that the typical use of DDI is within controlled environments: as the data is itself so often highly confidential, the metadata is often maintained and used within closed systems, except in those cases where it is exposed for discovery purposes, typically on websites. DDI has been used on a large scale: three excellent examples are its use within the *Consortium of European Social Science Data Archives (CESSDA)*;[9] its use by the *International Household Survey Network (IHSN)* community,[10] made up of more than 90 statistical agencies in the developing world; and its use by the largest SBE data archive in the US, the *Inter-university Consortium for Political and Social Research (ICPSR)*;[11] but there are many other examples.

The international standards organization DDI Alliance hosts a comprehensive list of projects using the DDI.[12] Academic users include the *UK Data Archive*[13] at the University of Essex, the *DataVerse Network*[14] at the Harvard-MIT Data Center, and *ICPSR* at the University of Michigan. Official data producers in more than 50 countries include the *Australian Bureau of Statistics*[15] and many national statistical institutes of the *Accelerated Data Program*[16] for developing countries. Examples for international organizations

[9] http://www.cessda.net
[10] http://www.ihsn.org
[11] https://www.icpsr.umich.edu
[12] http://www.ddialliance.org/ddi-adopters
[13] http://www.data-archive.ac.uk
[14] http://dataverse.org
[15] http://abs.gov.au
[16] http://adp.ihsn.org

are *UNICEF*,[17] the *Multiple Indicator Cluster Surveys*,[18] *The World Bank*,[19] and *The Global Fund to Fight AIDS, Tuberculosis and Malaria*.[20] [129] introduces DDI to those coming from national statistics institutes, as there is a large amount of information regarding DDI available today and sometimes it is difficult to know where to start and much of it comes from domains which are not familiar to those working with official statistics.

Data Lifecycle

DDI supports the entire research data life cycle and is specified using XML Schemas - organized in multiple modules corresponding to the individual stages of the data lifecycle. Common understanding is that both statistical data and metadata is part of a data lifecycle (see Figure 3.3). Data documentation is a process, not an end condition where a final status of the data is documented. Rather, metadata production should begin early in a project and should be done when it happens. The metadata can then be reused along the data lifecycle. Such practices incorporate documenting as part of the research method [168]. A paradigm change is enabled: on the basis of the metadata, it becomes possible to drive processes and generate items like questionnaires, statistical command files, and web documentation, if metadata creation is started at the design stage of a study in a well-defined and structured way. Multiple institutions are involved in the data lifecycle which is an interactive process with multiple feedback loops.

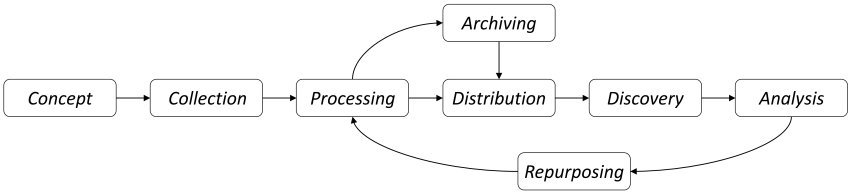

Fig. 3.3. Data Lifecycle

Relationships to other Metadata Standards

The *Dublin Core Metadata Element Set* [104] allows for the capture and expression of native Dublin Core elements, used either as references or as descriptions of a particular set of metadata. This is used for citation of the

[17] http://www.unicef.org
[18] http://www.unicef.org/statistics/index_24302.html
[19] http://www.worldbank.org
[20] http://www.theglobalfund.org

data, parts of the data documentation, and external material in addition to the richer, native means of DDI. The *DCMI Metadata Terms* [103] have been applied if suitable for representing basic information about publishing objects on the Web as well as for has part relationships. This approach supports applications which understand Dublin Core, but which do not understand DDI.

DDI is aligned with other metadata standards as well: with the *Statistical Data and Metadata eXchange (SDMX)* [161, 290] for exchanging aggregated data, *ISO/IEC 11179 (metadata registry)* [165] for building data registries such as question, variable, and concept banks, *ISO 19115* [162] for supporting users of geographic information systems, and the *PREMIS Data Dictionary for Preservation Metadata* [259] to support the preservation of digital objects and ensure their long-term usability.

3.2.2 Overview of DDI Specifications

In the following, we give an overview of what the DDI Alliance work products[21] are, what purpose they serve, and also address the coming developments. The overarching aim of the DDI Alliance is to create an international standard for describing data from the SBE sciences. Since its inception in 1995, the development of the DDI Alliance work products has moved in step with the evolving needs of the user communities seeking to document the data and the processes around its collection. [127, 194, 265, 314, 315] describe the foundations of the DDI metadata specifications in detail.

The work products have also evolved alongside developments in technology. XML, in which the current work products are expressed, has itself changed and developed. The standards to which the work products are aligned such as Dublin Core and most recently the Generic Statistical Information Model (GSIM) [199], have also evolved and the technologies available to implement the work products have matured. Current work products include:

- *DDI-Codebook* [93] is an XML structure for describing codebooks or data dictionaries, for a single study.
- *DDI-Lifecycle* [94], which is also expressed in XML, expands on the coverage of a single study along the data lifecycle and can describe several waves of data collection. It is very useful when dealing with serial data collection as is often seen in data production within statistical offices and long-standing research projects.
- *RDF Vocabularies*[22] currently cover three areas: one for describing data sets for the purposes of discovery on the Web (DDI-RDF), one for describing the structures of individual data sets (PHDD), and one for describing statistical classifications (XKOS). These vocabularies are based on the DDI-Codebook and DDI-Lifecycle XML structures, but are identical to neither. They are designed to be used in combination, and as such are non-duplicative (see Section 3.3).

[21] http://www.ddialliance.org/work-products-of-ddi-alliance
[22] http://www.ddialliance.org/Specification/RDF

- *DDI-Lifecycle MD (Model-Driven)* [96] is the draft which is currently under development. The further development of the DDI standard itself is model-driven in order to solve the problems associated with the XML Schema centric development of standard specifications. The model-driven development enables to break the model down to diverse concrete serializations such as RDFS/OWL, XML Schema, relational database schemas, and Java libraries.

3.2.3 DDI-Codebook

DDI-Codebook [93] is the original work product of the DDI Alliance. DDI-Codebook is an XML structure for describing codebooks or data dictionaries for a single study, i.e., a single wave of data collection. DDI-Codebook is designed to be used after-the-fact: it assumes that a data file exists as the result of a collection, and is then described using the XML structure provided by the specification.

DDI-Codebook is heavily used, being the structure behind such tools as *Nesstar*,[23] a system for the dissemination of statistical information and related documentation which is used by data archives within CESSDA. The largest single user-base is probably the IHSN which provides tools for documenting studies conducted by statistical agencies in the developing world, some of which are based on Nesstar.

3.2.4 DDI-Lifecycle

DDI-Lifecycle [94] is the result of a more demanding set of requirements emerging from the use of DDI-Codebook. It is again a complex metadata structure expressed using XML, but it is designed for different purposes. DDI-Lifecycle is capable of describing not only the results of data collection, but describing the metadata throughout the data collection process, from the initial conceptualization to the archiving of the resulting data. It can describe several versions of the data and metadata as they change across the data lifecycle. DDI-Lifecycle can describe several waves of data collection. It is very useful when dealing with serial data collection as is often seen in data production within statistical offices and long-standing research projects.

Any metadata which can be expressed using DDI-Codebook can also be described using DDI-Lifecycle which includes after-the-fact data description for single studies, as a necessary part of its broader scope. DDI-Lifecycle has - because of the purpose it was designed to serve - a more complex structure than DDI-Codebook.

DDI-Lifecycle can also be used to document specific sets of metadata outside of the description of a single study or set of studies. For example, areas of commonly shared metadata such as concepts, statistical classification, or geographic structures can be described and referenced by any number of studies.

[23] www.nesstar.com

Processing activities can be described and used to support a metadata-driven approach to the collection, processing, and publication of data.

3.2.5 DDI-Lifecycle MD

In the same way that experiences with DDI-Codebook have led to a broader set of requirements fulfilled by DDI-Lifecycle, implementations of the DDI-RDF vocabulary and DDI-Lifecycle have provided new requirements which will be met by *DDI-Lifecycle MD (Model-Driven)* [96].

DDI-Lifecycle MD will be very different from other DDI products, because it will be based on an information model of the metadata content. This model will be available for implementation in standard RDFS/OWL vocabularies and standard XML Schema structures, which will be equivalent. This form of model-based standard is a best practice in the world of standardization, and several other standards in other domains have been using this approach for many years.

DDI-Lifecycle MD will be another type of DDI work product: like DDI-Codebook and DDI-Lifecycle, it will be a standard structure for metadata related to the lifecycle of data in all stages, and thus will encompass much of what is available in those XML standards today. It will also include the functionality of DDI-RDF, since DDI-RDF is based on the structures of DDI-Codebook and DDI-Lifecycle. All items of DDI-RDF will have a mapping to elements in DDI-Lifecycle which in turn will be mapped to data model components in DDI-Lifecycle MD. In addition, DDI-Lifecycle MD allows to describe different types of data collection like surveys and register data and enables a broader coverage in terms of methodology and processes.

As the DDI Alliance anticipates that there may be many different implementation technologies using DDI as a basis, having an explicit model expressed using UML 2, will be of benefit. To provide an example, the SDMX standard is model-based and has standard implementations in XML Schemas, JSON [107], and EDIFACT. Additionally, the model was used as the basis for QB. Thus, moving DDI to a model-based orientation is a way of protecting the standard against technological change, and a way of guaranteeing alignment across different technology implementations.

The model-driven further development of the DDI standard enables to break the model down to diverse concrete serializations such as RDFS/OWL, XML Schema, relational database schemas, and Java libraries which can be used to develop own software applications [26]. These bindings may be adapted according to individual needs. In order to generate these bindings automatically, we transform for each binding the platform-independent model into a platform-specific model. Both types of models are physically stored in the standard data exchange format *XML Metadata Interchange (XMI)* [249] and validations rules are generated automatically out of individual platform-specific models in form of OWL axioms.

The development of the conceptual models of DDI-Codebook and DDI-Lifecycle was XML Schema centric. The further development of the DDI standard, however, is model-driven, as the XML Schema centric development of standards' specifications is associated with many drawbacks. A model is better maintainable and sustainable than XML Schemas as there are less side effects when changing the model. It is hard to maintain consistency between diverse XML Schemas which is much easier to ensure within a model. The model can also be shown to non-technical persons in form of UML class diagrams and further textual descriptions to adequately describe concepts and relationships among them.

DDI-Lifecycle MD supports the use of subsets of the whole model for a variety of purposes. These subsets are called *functional views*, are aligned with specific use cases, and therefore enable the use of DDI for different audiences, hides complexity of the whole model, and supports interoperability. In the end, the goal is that DDI-Lifecycle MD can present DDI to users in chunks, so users can learn and own just the parts of DDI that they need. The concept behind these functional views is based on views needed to manage the inherent complexity of large, software-intensive systems. It is commonly agreed that an architectural description consists of multiple views. Each view describes the architecture from the perspective of particular stakeholder concerns. The usage of views in architectural specification, analysis, and evolution is delineated by [57, 83, 146–150]. For each functional view and based on the model and additional rules, constraints will be generated to validate instance documents against them.

DDI-Lifecycle MD aligns DDI with the *Generic Statistical Information Model (GSIM)* [199], a reference model describing statistical information objects and processes. GSIM provides a common terminology across and between statistical organizations to maintain consistency and allows statistical organizations to understand and map common statistical information and processes [198]. As there are clear synergies between DDI, SDMX, and GSIM [307], DDI-Lifecycle MD serves as implementation layer for GSIM. Thereby, we build on experiences made by [174, 208, 214, 276] when implementing GSIM in their statistical organizations.

3.3 Representing Metadata on Research Data as Linked Data

The *DDI-RDF Discovery Vocabulary (DDI-RDF)*[24] [42] is intended to provide means to describe SBE sciences research data by essential metadata to support the discovery of unit-record data sets and related metadata using RDF technologies in the Web of Linked Data, which is the reason why *disco* is set as the namespace prefix for the vocabulary.

[24] http://www.ddialliance.org/Specification/RDF/Discovery

Many archives and other organizations have large amounts of research data, sometimes publicly available, but often confidential in nature, requiring applications for access. Such organizations use the proven and highly detailed DDI standard for describing data sets of this type. When we consider how such a standard could be used as a basis for an RDF vocabulary, we realize that the requirements are very different. The most obvious use case is that of discovery, given that much of the data is highly confidential, and that access to the data must be applied for in most cases. The challenges of searching the Web of Linked Data are enormous – the sheer range of information is incredibly broad. Thus, the almost twelve hundred of metadata fields within DDI is itself a problem. The DDI model must be significantly reduced in complexity to be meaningful to cover these requirements. The fact that DDI is not specific to any particular research domain or type of research data is a positive feature, however, as the range of data to be exposed into the Web of Linked Data is also very broad.

We make use of the DDI standard to create a simplified version of this model by choosing the DDI elements which are needed for the discovery purpose. With the background of the broadness and complexity of DDI, DDI-RDF focuses on a small subset of DDI. The selection relies on use cases which are oriented on the discovery of data in the Linked Data context and possible usage within the Web of Linked Data. DDI-RDF has emerged as a massive simplification of the DDI standard, optimized for querying using Semantic Web technologies such as SPARQL.

Part of this best practice is the reuse of existing vocabularies wherever possible, and the extension of existing vocabularies where needs are almost, but not completely covered. Widely accepted and adopted vocabularies are reused to a large extend, as there are features of the DDI which can be addressed through other vocabularies, such as: (1) describing metadata for citation purposes using Dublin Core, (2) describing aggregated data like multi-dimensional tables using QB, and (3) delineating code lists, category schemes, mappings between them, and concepts like topics using SKOS.

In designing this vocabulary, every attempt has been made to meet the requirements of the different technologies and needs found in the Linked Data world [144], as opposed to the world of data archives, research institutes, data libraries, and government statisticians [168]. We look at the motivations of individuals from two different communities separately, because while they are complementary, they are very different.

Motivation for Data Professionals and the DDI Community

For data professionals, the use of the data they produce or disseminate is often a primary goal. For those working in data archives and data libraries, the service they offer is access to research data for secondary use, and an increase in the use of their data is perceived as a positive sign.

For government statisticians, it is the same – they produce the "official" data to support policy, and they perceive the use of their data as a contribution to society, and a fulfillment of their mission. [128] provides a description of how the collaboration between open government initiatives, the Linked Data community, and experts in the field of statistics and research data could enormously improve the usability of quantitative government data on the Web. DDI-RDF provides a useful model for describing some of the data sets now being published by open government initiatives, by providing a rich metadata structure for them. While the data sets may be available (typically as CSV files), the metadata which accompanies them is not necessarily coherent, making the discovery of these data sets difficult. This vocabulary helps to overcome this difficulty by allowing for the creation of standard queries to programmatically identify data sets, whether made available by governments or held within data archives.

For researchers, the reuse of the data they collect is something which enhances their reputation and career, through an emerging system of data citation which is very similar to the traditional citation of research papers. Thus, the various members of the DDI community are very interested in having their data be discovered and used.

This is somewhat problematic, however – it is not enough simply to publish data to the Web, which is very often illegal for reasons of privacy and confidentiality. Instead, a researcher looking for unit-record data is often required to apply for access, and to make commitments about how the data will be used and released. These issues are taken very seriously by data professionals for a variety of reasons: first, if people asked to fill out a survey do not trust the person administering the survey, they will refuse to respond, making the collection of good raw data with surveys difficult or impossible. Thus, researchers want to be trusted by the people they study. Additionally, the release of confidential information is illegal and potentially very destructive, and can result in prosecution.

The degree of "statistical literacy" among users is always a major concern with those who work at data libraries and archives, supporting researchers. When using raw data and unit-record data, there is a significant set of skills which is required to produce valid research. These skills require access to the best possible metadata about the data. This is especially true when working with data coming from different sources, something which researchers are often very keen to do.

Thus, the DDI community is torn in two directions: on the one hand, they very much want people to use their data, and hence are very interested in advertising their data through the Web of Linked Data; on the other hand – and especially after seeing the metadata-free nature of many data sets published at open data sites such as data.gov in the US – they are concerned at the lack of standard, highly-detailed metadata which is required for the correct analysis and use of unit-record data.

Ultimately, the DDI-based RDF vocabulary is done as a way of making sure that when unit-record (or raw) data is published in the Web of Linked Data, this will be done in a way which allows for correct and responsible use of that data. The basic idea here is to reap the benefits of broader use of existing data resources, while benefitting from the knowledge and experience of working with data which is the hallmark of the DDI community and its members.

Motivation for the Linked Data Community

From the perspective of the Linked Data community, the benefit is a simple one – to put all of the data holdings of data archives and statistical organizations into the Web of Linked Data. DDI-RDF encourages the holders of the data to be confident that sufficient metadata is being published to permit discovery and use of the data.

The benefits for the Linked Data community lie at hand, as there was no such vocabulary available with a comparable level of detail for representing complex entities and relations regarding the complete lifecycle of research data as DDI-RDF provides. The publication of research data in the Web of Data became popular and important in various domains beside the SBE sciences, so that a valuable contribution can be seen in the introduction of DDI-RDF.

RDF-based tools are able to take advantage of this publication, without requiring the use of the complicated XML schemas which most DDI implementations require. Additionally, data sets described using this vocabulary can be easily linked with other data sets, and can be more easily connected to related web-based descriptions, making the data and the results of research more closely connected. Further, the possibility exists of making explicit the metadata around published, but under-documented data sets from open government initiatives, in a standard and understood form, by organizations other than those which published the data sets themselves.

3.3.1 Overview of the Conceptual Model

Figure 3.4 gives an overview of the conceptual model of DDI-RDF - more detailed descriptions of all classes and properties as well as various examples are given in the DDI-RDF specification [42].

Identification and Relations to DDI-XML

In DDI, a lot of entities hold particular identifiers such as for different DDI versions of DDI-Codebook or DDI-Lifecycle, but also persistent identifiers for persons or organizations that are encoded in a particular identifier scheme like ORCID[25] or FundRef.[26] In general, such identifiers can be added to each

[25] http://orcid.org
[26] http://www.crossref.org/fundref

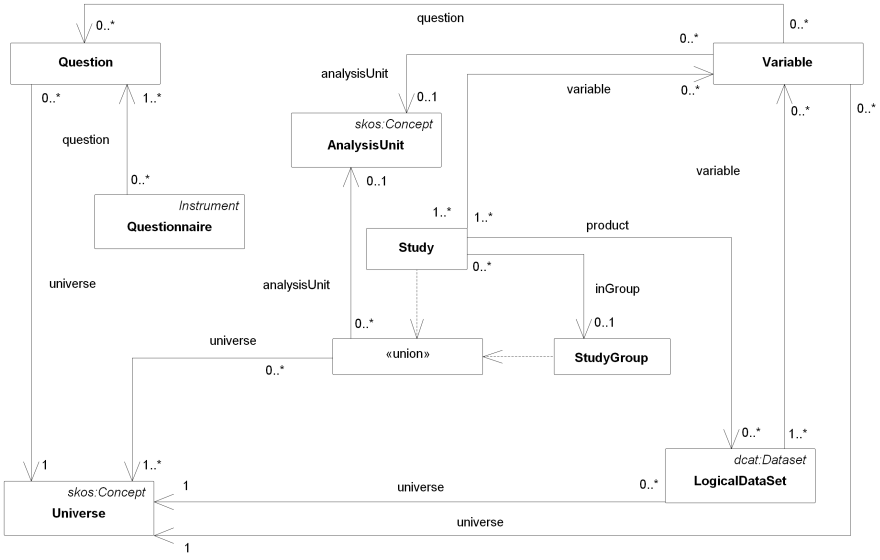

Fig. 3.4. Overview of the Conceptual Model

entitiy in DDI-RDF, since every entity is defined to be an *rdfs:Resource*. General metadata elements which can be used on every resource include *skos:prefLabel* to indicate the preferred label of this element. Each resource must have (*adms:identifier*) one or multiple identifiers (*adms:Identifier*). Resources of the class *adms:Identifier* can include the actual identifier itself as well as information about the identifier scheme, version, and agency.

Since DDI-RDF only covers a small subset of the DDI-XML specifications DDI-Codebook and DDI-Lifecycle, it is worthwhile to have a relationship to original DDI-XML files. Unique identifiers for specific DDI versions are used for easing the linkage between DDI-RDF metadata and the original DDI-XML files. Every DDI-RDF resource can be related (*disco:ddifile*) to a *foaf:Document* representing a DDI-XML file containing further descriptions.

Versioning, Access Rights, and Licensing

Any resource can have versioning information (*owl:versionInfo*). The most typical cases, however, are the versioning of (1) the whole metadata, i.e., the RDF file, (2) studies, as a study goes through the life cycle from conception through data collection, and (3) data files. Data sets (*disco:LogicalDataSet*) may have access rights statements (*dcterms:accessRights*) and licensing information (*dcterms:license*) attached to it (see Figure 3.5).

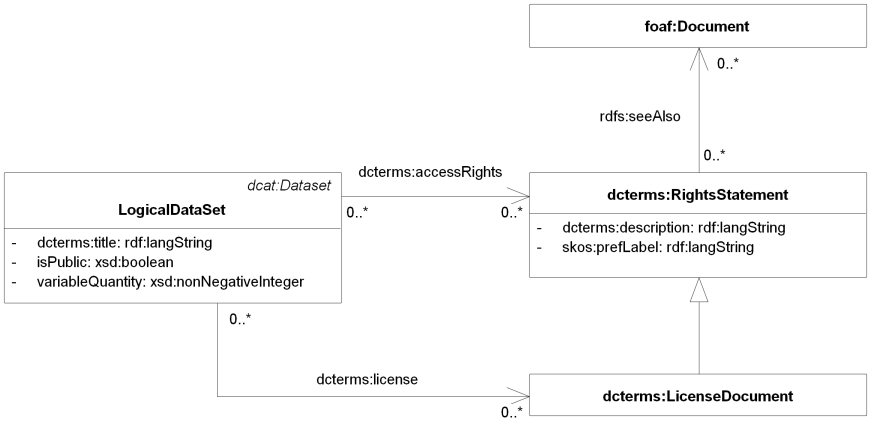

Fig. 3.5. Access Rights Statements and Licensing Information

Coverage

Studies, series, data sets, and data files may have a spatial (*dcterms:spatial* pointing to *dcterms:Location*), temporal (*dcterms:temporal*), and topical (*dcterms:subject*) coverage (see Figure 3.6 for the coverage of studies and series).

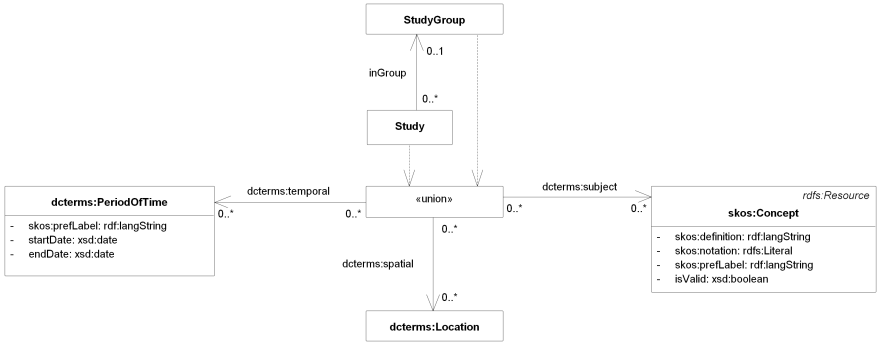

Fig. 3.6. Coverage

For stating temporal coverages, *dcterms:temporal* is used pointing to *dcterms:PeriodOfTime*. For time periods, labels can be attached (*skos:prefLabel*) and start (*disco:startDate*) and end dates (*disco:endDate*) can be defined. Topical coverages can be expressed using *dcterms:subject*. DDI-RDF foresees the use of *skos:Concept* for the description of topical coverages.

Studies and Series

To understand DDI-RDF, there are a few central classes, which can serve as entry points. The first of these is *disco:Study* representing the process by which a data set was generated or collected. A simple study supports the stages of the full data lifecycle in a modular manner. In some cases, where data collection is cyclic or on-going, data sets may be released as a *series* (*disco:StudyGroup*), where each cycle or wave of the data collection activity produces one or more data sets. This is typical for longitudinal studies, panel studies, and other types of series. In this case, a number of disco:Study objects would be collected (*disco:inGroup*) into a single disco:StudyGroup.

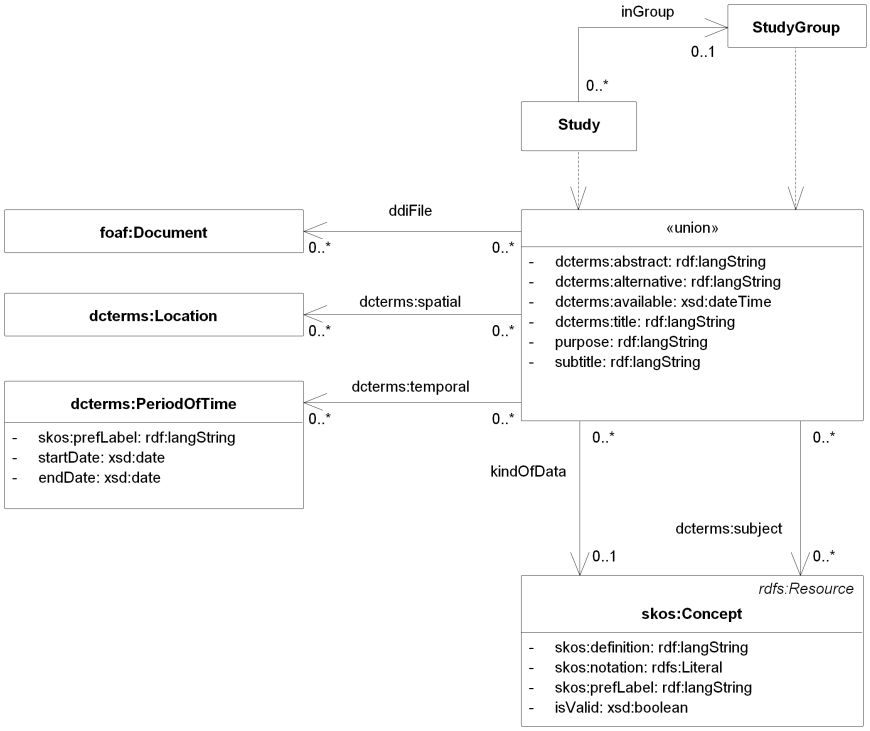

Fig. 3.7. Studies and Series

Data properties for studies and series (see Figure 3.7) include information about funding, organizational affiliation, creator, contributor, publisher, abstract, title, purpose, and information about the date and the time since when the study/series is publicly available. The property *disco:kindOfData* describes, with a string or a term from a controlled vocabulary, the kind of data documented in the data sets of a study. Examples include survey, census, administrative, measurement, assessment, demographic, and voting data.

Data is collected about a specific phenomenon, typically involving some target population of a defined class of people, objects, or events (*disco:Universe*), and focusing on the analysis of a particular type of subject (*disco:AnalysisUnit*). If, for example, the adult population of Finland is being studied, the analysis unit would be individuals or persons.

Logical Data Sets and Data Files

Data sets have two representations: (1) a logical representation which describes the contents of the data set and (2) a physical representation which is a distributed file holding that data. A *disco:LogicalDataSet* (see Figure 3.8), an extension of *dcat:Dataset*, describes the content of the file, i.e., its organization into a set of variables.

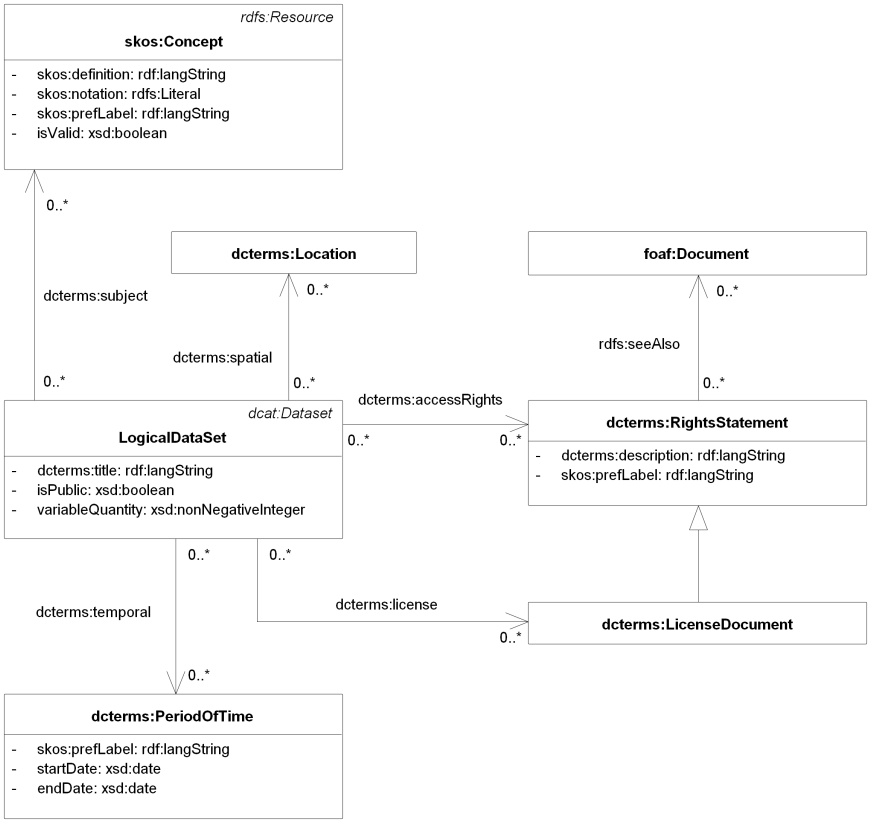

Fig. 3.8. Logical Data Sets

For logical data sets, one can state a title (*dcterms:title*) and a flag indicating if the unit-record data set is publicly available (*disco:isPublic*). The

data property *disco:variableQuantity* on data files and logical data sets is useful to have (1) when no variable level information is available or (2) when only a stub of the RDF is requested, e.g., when returning basic information on a study we do not need to return information on potentially hundreds or thousands of variables.

Physical, distributed files containing unit-record data sets are represented by *disco:DataFile* (see Figure 3.9), an extension of *dcat:Distribution* and *dcterms:Dataset*. It is possible to format (*dcterms:format*) data files in many different ways, even if the logical content is the same. Data files can be described (*dcterms:description*) and case quantities (*disco:caseQuantity*), versions (*owl:versionInfo*), and provenance information (*dcterms:provenance*) can be stated.

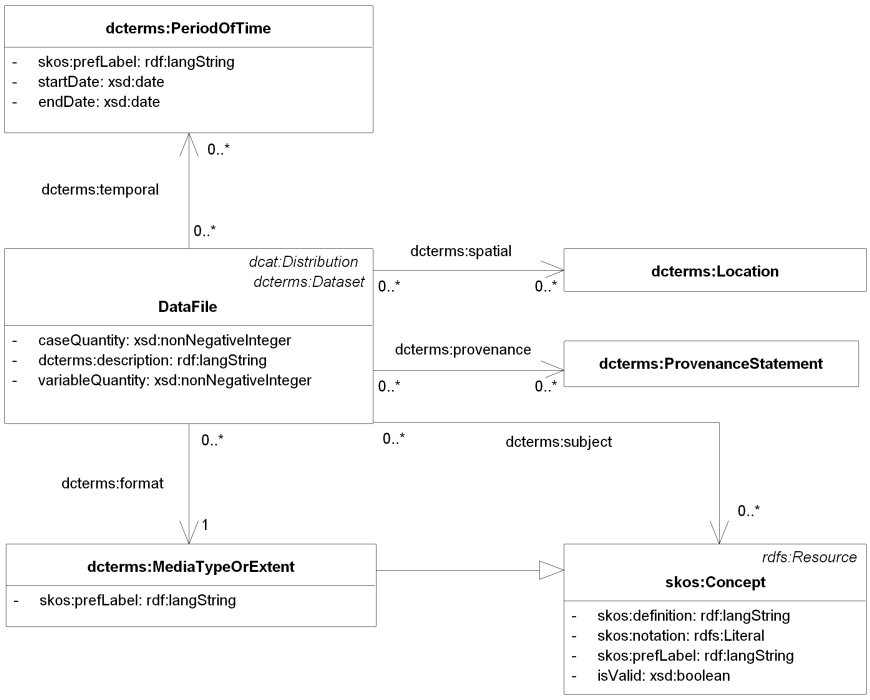

Fig. 3.9. Data Files

Descriptive Statistics and Relations to Aggregated Data

An overview of the unit-record data can be given either by descriptive statistics or aggregated data. *disco:DescriptiveStatistics* may be minimum, maximum, mean values, and standard deviations of specific variables, or absolute

and relative frequencies of particular codes. *disco:SummaryStatistics* pointing to variables and *disco:CategoryStatistics* pointing to codes are both descriptive statistics (see Figure 3.10). Available category statistics types are *frequency*, *percentage*, and *cumulative percentage*. Available summary statistics types are skos:Concepts and organized in a controlled vocabulary.

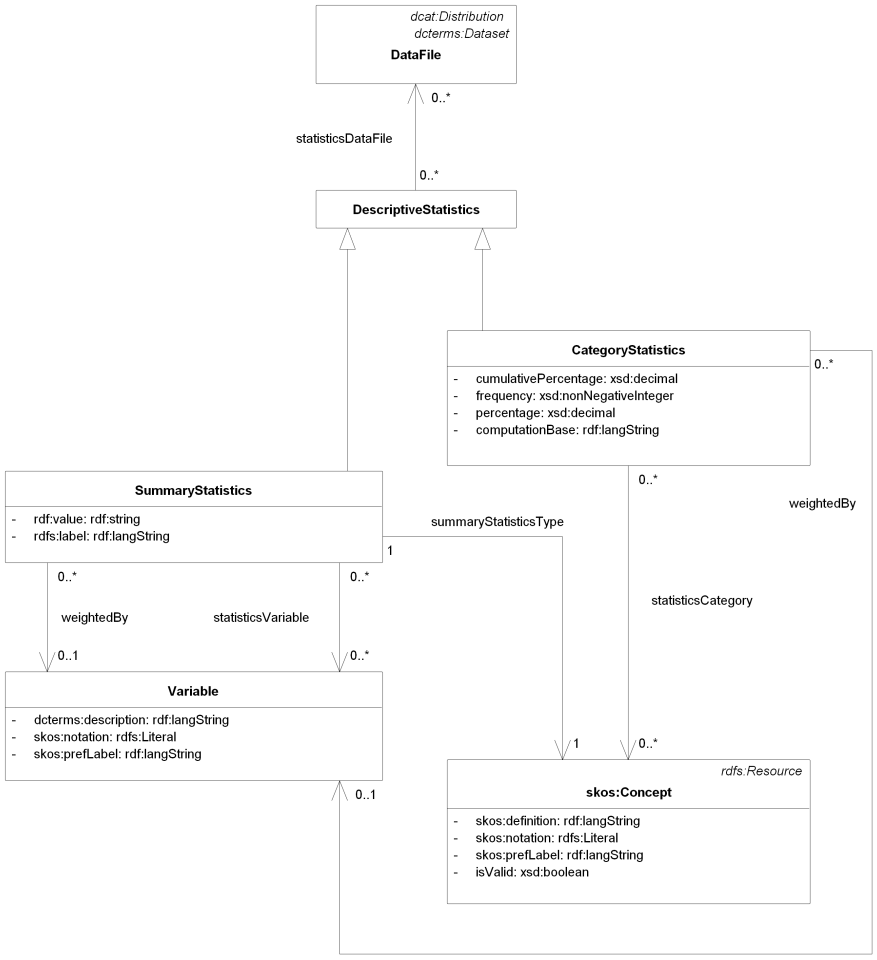

Fig. 3.10. Descriptive Statistics

A *qb:DataSet* represents aggregated data such as multi-dimensional tables. Aggregated data is derived from (*disco:aggregation*$^-$) unit-record data by statistics on groups, or aggregates such as counts, means, or frequencies.

Variables

When it comes to understanding the contents of a data set, this is done using the *disco:Variable* class (see Figure 3.11). Variables (e.g., *age* or *sex*) provide a definition of the column in a rectangular data file, and can associate it with a theoretical concept (*skos:Concept*) to be investigated (e.g., *education*) and a *disco:Question* (e.g., 'How old are you?') - the question in a *disco:Questionnaire* which was used to collect the data. We use *skos:narrower* and *skos:broader* to build hierarchies of theoretical concepts within a *skos:ConceptScheme* of a particular study or series.

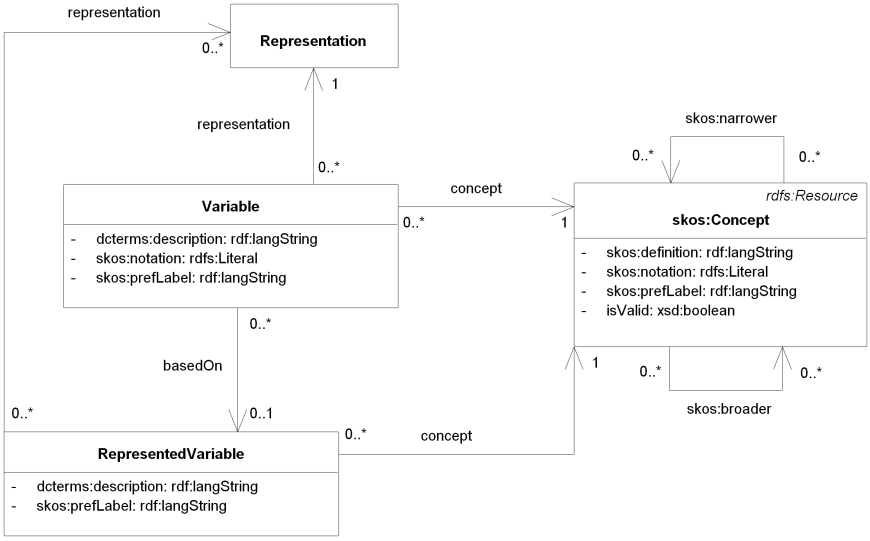

Fig. 3.11. Variables and Represented Variables

A *variable* is a characteristic of a unit being observed and might be the answer of a question, have an administrative source, or be derived from other variables (e.g., *age group* derived from *age*). Variables can be further described (*dcterms:description*), *skos:notation* is used to associate names with variables and labels can be assigned to variables via *skos:prefLabel*. Variables may be based on represented variables (*disco:RepresentedVariable*) encompassing study-independent, re-usable parts of variables like the classification of occupations.

Representations

Variables, represented variables, and questions may have representations (see Figure 3.12). A *disco:Representation* may be an ordered or unordered code

list, i.e., a set of codes (e.g., 0, 1, 2, ...) and categories (human-readable labels of codes), or a set of values of possibly multiple datatypes. The variable *age*, e.g., may be represented by values of the datatype *xsd:nonNegativeInteger* or by an ordered code list of individual ages or age clusters (e.g., '0 to 10' and '11 to 20').

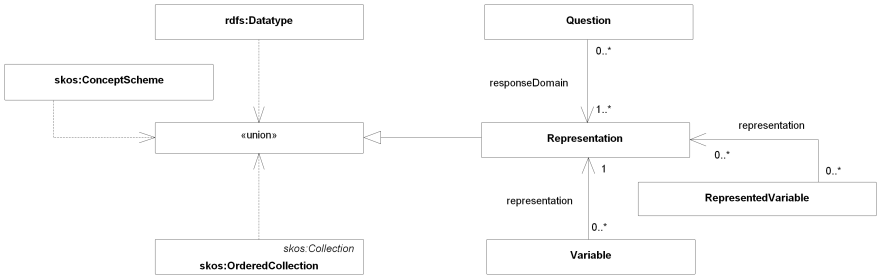

Fig. 3.12. Representations of Variables, Represented Variables, and Questions

Codes and categories themselves are represented using *skos:Concept* and organized in *skos:ConceptSchemes* in case of an unordered code list (see Figure 3.13). A *skos:OrderedCollection* organizes codes in a particular order within a *skos:memberList*. Theoretical concepts can be described using *skos:definition*, codes by *skos:notation*, and categories by *skos:prefLabel*.

Data Collection

The data collection produces the data sets of a study. In some cases, where data collection is cyclic or on-going, data sets may be released as a series, where each cycle or wave of the data collection activity produces one or more data sets. The data for the study is collected by an *instrument* (see Figure 3.14), i.e., a questionnaire, an interview, or another entity used as a means of data collection.

The purpose of an instrument is, in the case of a study, to record the flow of a *questionnaire* and its use of questions. A *question* is designed to get information from a respondent upon a subject or a sequence of subjects. Instruments can be described, may be associated with human-readable labels, and may have external documentations. A question has a question text (e.g., 'How old are you?'), a label (e.g., *age*), and a response domain, the representation of the question (e.g., the datatype *xsd:nonNegativeInteger*).

3.3.2 Reuse of and Relations to other Vocabularies

Widely accepted and adopted vocabularies are reused within the conceptual model of DDI-RDF to a large extend. There are features of DDI which can

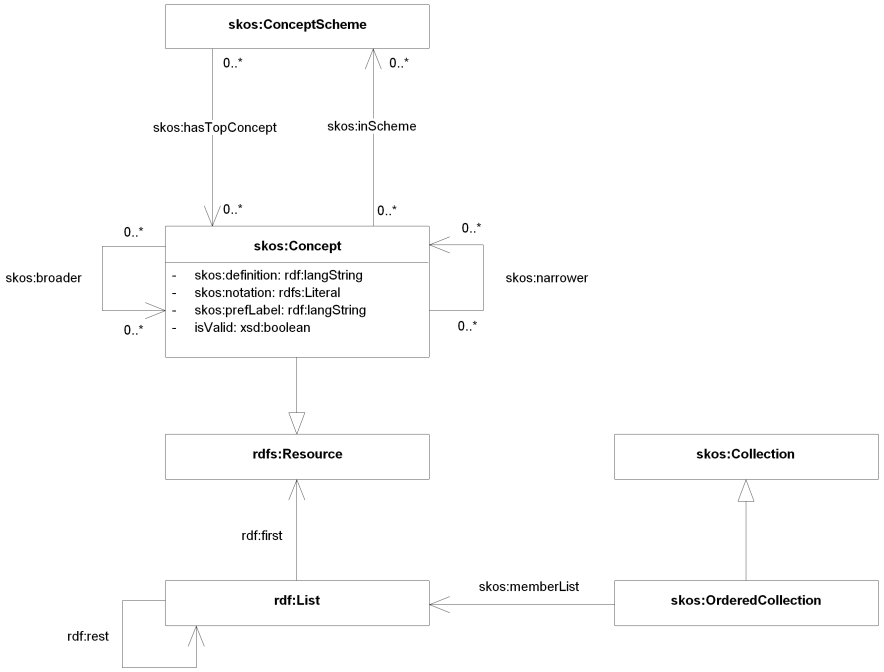

Fig. 3.13. Organization of Code Lists

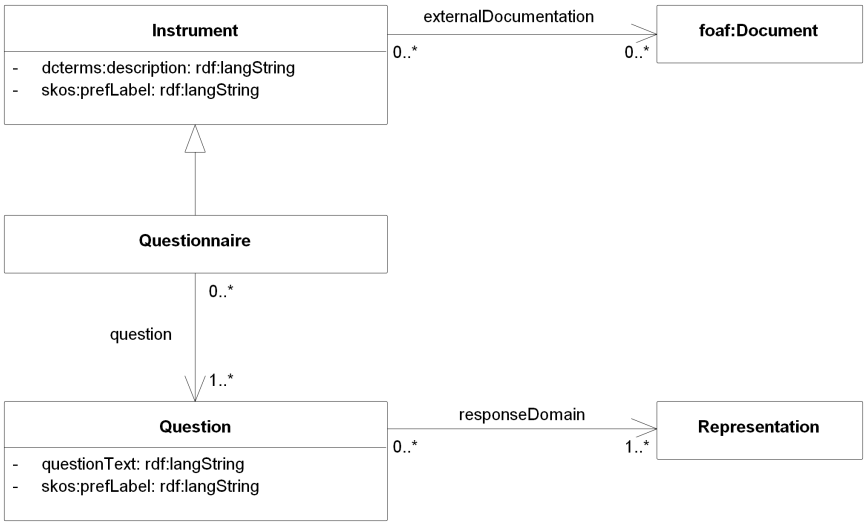

Fig. 3.14. Data Collection

be addressed through other vocabularies, such as detailed provenance information on research data and its metadata (*PROV-O*), catalogs of data sets (*DCAT*), aggregated data (*QB*), formal statistical classifications (*XKOS*), code and category lists, theoretical concepts, and concepts like topics (*SKOS*), persistent identifiers (*ADMS*), and arbitrary objects, processes, and their attributes (*SIO*). Furthermore, we reuse the *Vocabulary of Interlinked Datasets (VoID)* [2, 90] for expressing general metadata about data sets, *Friend of a Friend (FOAF)* [62] to describe person-level data, the *Organization Ontology (ORG)* [269] to model organization related information, and the *DCMI Metadata Terms* [103] to describe general metadata of DDI-RDF constructs for citation purposes.

The *PROV Ontology (PROV-O)* [122, 203, 226] is used to represent detailed provenance information of research data and its metadata, to describe information on ownerships, and to interchange provenance information generated in different systems and under different contexts. Terms of PROV-O are organized among three main classes: *prov:Entity*, *prov:Activity* and *prov:Agent*. While classes of DDI-RDF can be represented either as entities or agents, particular processes for, e.g., creating, maintaining, and accessing data sets can be modeled as activities.

Properties like *prov:wasGeneratedBy*, *prov:hadPrimarySource*, *prov:wasInvalidatedBy*, or *prov:wasDerivedFrom* describe the relationship between classes for the generation of data in more detail. In order to link from a *disco:Study* to its original DDI-XML file, the property *prov:wasDerivedFrom* can be used. PROV-O allows for representing versioning information by using the terms *prov:Revision*, *prov:hadGeneration*, and *prov:hadUsage*. PROV-O can also be used to model information and relationships that are relevant for determining accuracy, quality, and comparability of a data set with others. By utilizing the properties *prov:qualifiedInfluence* or *prov:wasInformedBy*, qualified statements can be made about a relationship between entities and activities, e.g., that and how a particular method influenced a particular data collection or data preparation process.

Combining terms from DCAT and DDI-RDF can be useful for a number of reasons:

- Describing collections or catalogs of research data sets using DCAT
- Providing additional information about physical aspects of research data files (e.g., file size and file formats) using DCAT
- Offering information about the data collection that produced the data sets in a data catalog using DDI-RDF
- Providing information about the logical structure (e.g., variables and theoretical concepts) of tabular data sets in a data catalog using DDI-RDF

DCAT is better suited to describe collections and catalogues of data sets. DDI-RDF, on the other side, supports richer descriptions of groups of data sets or individual data sets. We map to DCAT in two places: the

disco:LogicalDataSet is an extension of the *dcat:Dataset* and physical, distributed data files are represented by *disco:DataFile* which is itself an extension of *dcat:Distribution*.

We reuse SKOS to a large extent to represent theoretical concepts and their hierarchies, unordered and ordered code lists, formats of data files, the kind of data documented in the data sets of a particular study, and coverages. Spatial, temporal, and topical coverages are directly attached to studies, logical data sets, and data files.

Especially persons and organizations may hold one or more persistent identifiers of particular schemes and agencies (e.g., ORCID[27] or FundRef[28]) that are not considered by the specific identifiers of DDI. To include those identifiers and for distinguishing between multiple identifiers for the same resource, we use the *Asset Description Metadata Schema (ADMS)* [9]. As a profile of DCAT, ADMS aims to describe semantic assets, i.e., reusable metadata and reference data. Individuals of the class *adms:Identifier* can be added to any rdfs:Resource by the property *adms:identifier* (see Figure 3.15). That identifier can contain properties that define the particular identifier itself, but also its scheme, version, and managing agency. Although utilized primarily for describing identifiers of persons and organizations, it is allowed to attach an adms:Identifier to instances of all classes in DDI-RDF.

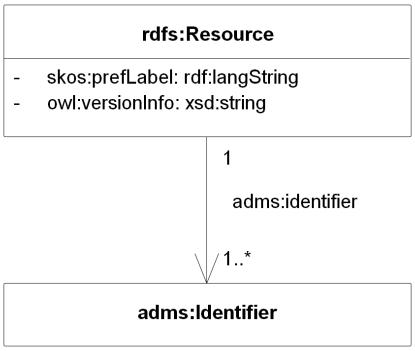

Fig. 3.15. Identification

The *Semanticscience Integrated Ontology (SIO)* [14, 106] is a simple ontology to describe arbitrary (i.e., real, hypothesized, virtual, and fictional) objects, processes, and their attributes. A *sio:SIO_000367 (variable)* represents a value that may change within the scope of a given operation or set of operations. For instance, in the context of mathematics or statistics, SIO variables are information content entities that can be used to indicate the independent, dependent, or control variables of a study or experiment. The similarity be-

[27] http://orcid.org
[28] http://www.crossref.org/fundref

tween a SIO variable and a disco:Variable is that they may be associated with a theoretical concept like *sex*, *age*, or *citizenship*. We see disco:Variable as being equivalent to the SIO variable class.

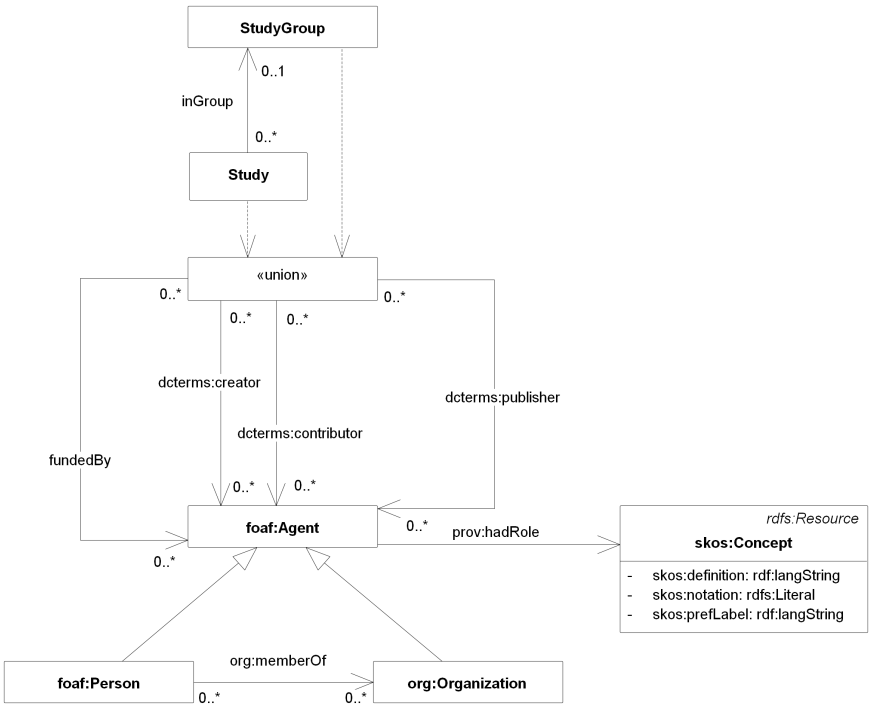

Fig. 3.16. Reuse of DCMI Metadata Terms, FOAF, and ORG

The DCMI Metadata Terms are used to describe general metadata of DDI-RDF constructs such as a study abstract (*dcterms:abstract*), a study or data set title (*dcterms:title*), a human readable description of DDI-RDF resources (*dcterms:description*), provenance information for a data file (*dcterms:provenance*), and the date (or date range) at which a study will become available (*dcterms:available*). Creators (*dcterms:creator*), contributors (*dcterms:contributor*), and publishers (*dcterms:publisher*) of studies and series are *foaf:Agents* which are either *foaf:Persons* or *org:Organizations* whose members are foaf:Persons. Studies and series may be funded by (*disco:fundedBy*) foaf:Agents, whereby disco:fundedBy is defined to be a sub-property of dcterms:contributor (see Figure 3.16).

3.3.3 Use Cases

In this section, we introduce several representative real world use cases that show (1) the usage of the DDI-RDF vocabulary, (2) its importance for data professionals, the SBE sciences, and the Linked Data community, and (3) the interaction between DDI-RDF and other vocabularies. Just like the use cases delineated by [50, 53, 193, 318], these use cases have in common that they represent real information needs of researchers. The DDI-RDF specification [42] contains additional example data which can be consulted to get details of how to construct DDI-RDF instance data and to get a feeling of the full potential of DDI-RDF to represent metadata on statistical data.

UC1: Enhancing the discovery of data by providing related metadata. Many archives and government organizations have large amounts of data, sometimes publicly available, but often confidential in nature, requiring applications for access. While the data sets may be available (typically as CSV files), the metadata which accompanies them is not necessarily coherent, making the discovery of these data sets difficult. A user has to read related documents to determine if the data is useful for the intended research purposes. The data provider could enhance the discovery of data by providing key metadata in a standardized form which would allow to create standard queries to programmatically identify needed data sets [40].

UC2: Searching for studies by free text search in study descriptions. The most natural way of searching for data is to formulate the information need by using free text terms and to match them against the most common metadata, like title, description, abstract, or unit of analysis. A researcher might search for relevant studies which have a particular title or keywords assigned to in order to further explore the data sets attached to them. The definition of an analysis unit might help to directly determine which data sets the researcher wants to download afterwards. A typical query could be 'find all studies with questions about commuting to work' [40, 318].

UC3: Searching for studies by publishing agency. Researchers are often aware of the organizations which disseminate the kind of data they want to use. This scenario shows how a researcher might wish to see the studies which are disseminated by a particular organization, so that the data sets which comprise them can be further explored and accessed. 'Show me all the studies for the period 2000 to 2010 which are disseminated by the UK Data Archive' is an example of a typical query [40, 318].

UC4: Searching for studies by coverage. Researchers often want to know which studies exist for a specific country (spatial coverage, e.g., France), time period (temporal coverage, e.g., 2005), and subject (topical coverage, e.g., election). The coverage in this example is separated by the three dimensions: country, time, and subject [41, 318].

UC5: Linking publications to related data sets. Publications, which describe ongoing research or its output based on research data, are typically held in bibliographical databases or information systems. By adding unique

and persistent identifiers to DDI-based metadata for data sets, these data sets become citable in research publications and thereby linkable and discoverable for users. On the other side, the extension of research data with links to relevant publications is also possible by adding citations and links. Such publications can directly describe study results in general or further information about specific details of a study, e.g., publications of methods or the design of the study or about theories behind the study [40, 41, 50]. [175] discuss another approach to link publications to data.

UC6: Searching for data sets by accessibility. This scenario describes how to retrieve data sets which fulfill particular access conditions. Many research data sets are not freely available, and access conditions may restrict some users from accessing some data sets. It is expected that the researcher looking for data might wish to see the data sets which are either publicly available or which meet specific access conditions or license terms. Access conditions vary by country and institution. Users may be familiar with the specific licenses which apply in their own context [40, 318].

UC7: Links to external thesauri. Theoretical concepts can be connected with questions, variables, or descriptive statistics to provide information about their topic. Such concepts are typically organized in concept schemes which are often similar to traditional thesauri or classification systems regarding their structure and content. When assigning concepts, either an existing concept scheme has to be used or a new one has to be defined. A precise annotation of such concepts is relevant for users when searching for studies, which, e.g., cover specific concepts or contain questions regarding a specific theme.

In a lot of cases, however, the user does not know which terms or classification systems have been used to provide these concepts. In such cases, mappings from concepts to terms of established thesauri or dictionaries like EuroVoc,[29] WordNet,[30] or LCSH[31] and more domain-specific thesauri such as the STW Thesaurus for Economics[32] or the Thesaurus for the Social Sciences (TheSoz)[33] can be helpful to recommend users suitable terms for search which are used in a study as concepts.

Such mappings between thesauri are a typical instrument for information retrieval. Therefore, it is quite reasonable to connect concepts to terms of existing knowledge systems to (1) describe these concepts and (2) provide information retrieval related services for users like search term recommendation during search. The inclusion of thesauri, which often provide an established and mature term corpora in their specific discipline, does not only disseminate the use of such vocabularies, but also the potential reuse of the concepts in other Linked Data applications [41, 50].

[29] http://eurovoc.europa.eu
[30] https://wordnet.princeton.edu
[31] http://id.loc.gov/authorities/subjects.html
[32] http://zbw.eu/stw/version/latest/about.en.html
[33] http://www.gesis.org/en/services/research/thesauri-und-klassifikationen/social-science-thesaurus

UC8: Searching for studies and data sets by persons and organizations. Researchers may search for studies and data sets by producer, creator, or contributor [56, 318]. A typical scenario within the SBE sciences could be: A researcher aggregates unit-record data sets of the European study EU-SILC[34] and acted on behalf of the research institute GESIS[35] which funded the study. Representative queries could be: (1) Which persons, working for the research institute GESIS, created the German General Social Survey (ALLBUS),[36] a particular group of studies in Germany? (2) Which organizations and persons contributed to the creation of the European study EU-SILC? (3) Which persistent identifiers are assigned to persons and organizations publishing the European study EU-LFS?[37]

UC9: Searching for data sets by statistical classification. Researchers may be interested in data sets having a specific statistical classification like ISCO [56]. One question typically asked by SBE researchers could be to query on the semantic relationships which are defined for concepts within statistical classifications using XKOS properties. By means of these properties not only hierarchical relations can be queried but also part of relationships (*xkos:hasPart*), more general (*xkos:generalizes*) and more specific (*xkos:specializes*) concepts, and positions of concepts in lists (*xkos:previous*, *xkos:next*).

UC10: Searching for data in data collections. While DDI-RDF and QB provide terms to describe unit-record (*disco:LogicalDataSet*) and aggregated data sets (*qb:DataSet*), DCAT enables to describe these data sets inside of data collections like repositories, catalogs, or archives. The relationship between data collections (*dcat:Catalog*) and their contained data sets (*dcat:Dataset*) is useful, since such collections are a typical entry point when searching for data (see Figure 3.17).

A search for data may consist of two phases. In a first phase, the user searches for different records (*dcat:CatalogRecord*) inside a data catalog. This search can differ according to the users' information needs. While it is possible to search for metadata provided inside such a record (e.g., *dcterms:title*, *dcterms:description*), the user may also search for more detailed information about the data set or its distribution (*dcta:Distribution*, *disco:DataFile*). A user may want to search for data sets of a particular topical (*dcterms:subject*, *dcat:theme*, *dcat:keyword*), temporal (*dcterms:temporal*), and spatial (*dcterms:spatial*) coverage, or for certain formats (*dcterms:format*) in which a distribution of the data is available. Data sets of data catalogs are described by themes they cover. Since these themes are organized in a taxonomy, they can be used for an overall search in all data sets of a data catalog. As the search of the first phase may result in multiple hits of data sets, another more sophisticated search has to be executed in a second phase to find out which

[34] http://www.gesis.org/en/missy/metadata/EU-SILC
[35] http://www.gesis.org
[36] http://www.gesis.org/en/allbus
[37] http://www.gesis.org/en/missy/metadata/EU-LFS

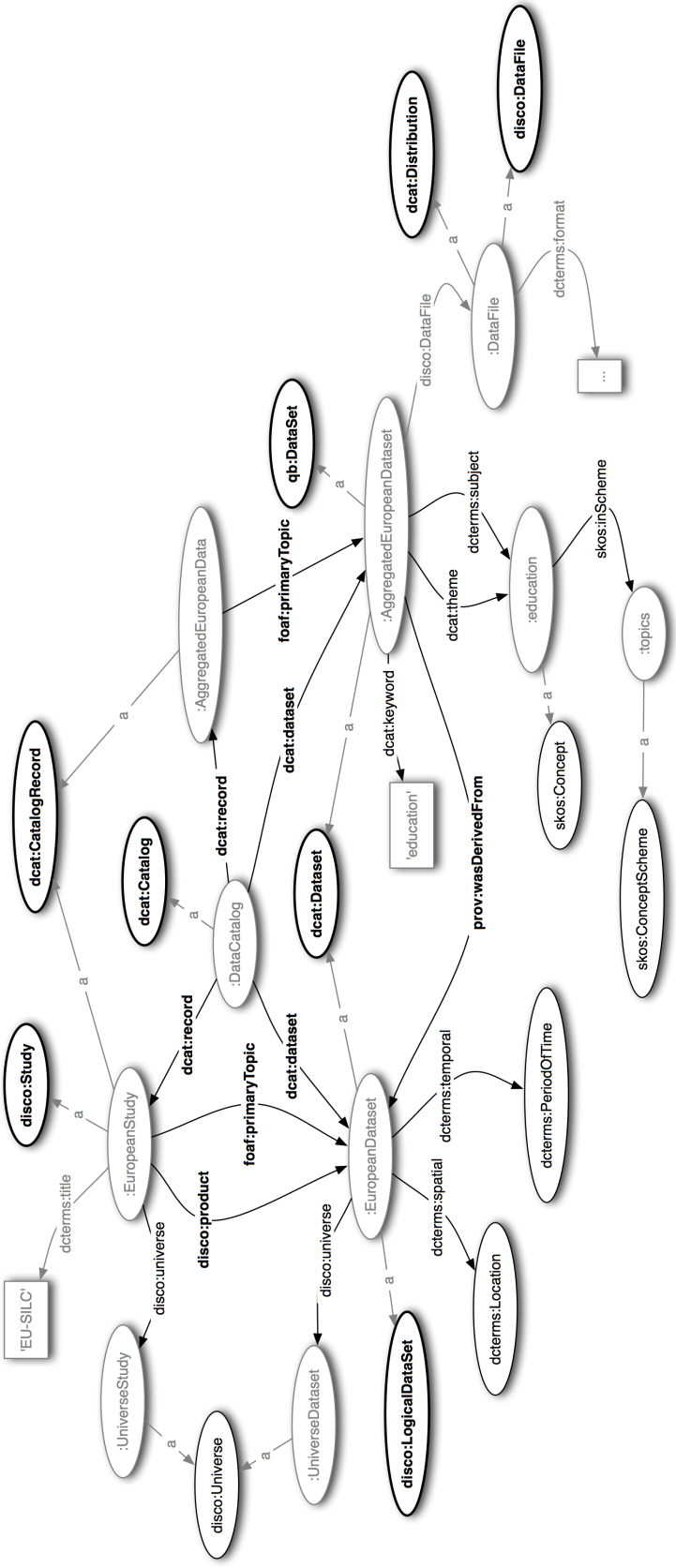

Fig. 3.17. Searching for Data in Data Collections

data sets are relevant for the user. In case the user finds aggregated data sets published in QB, the property *prov:wasDerivedFrom* may hold a link to discover the original unit-record data sets represented in DDI-RDF [56, 318].

3.3.4 Implementations

We implemented a direct mapping between DDI-RDF and DDI-XML (DDI-Codebook and DDI-Lifecycle) using XSLT stylesheets,[38] which can be used as a reference implementation showing how elements in DDI-Codebook and DDI-Lifecycle are mapped to DDI-RDF and back.

The structure of DDI-Codebook differs substantially from DDI- Lifecycle. DDI-Codebook is designed to describe metadata for archival purposes and its structure is very predictable and focused on describing variables with the option to add annotations for question texts. DDI-Lifecycle, on the other hand, is designed to capture metadata from the early stages in the research process. DDI-Lifecycle enables capturing and reuse of metadata through referencing: a lot of the metadata can be described in modules and references are used to a large extend, e.g., between questions and variables. DDI-Lifecycle has more elements and is able to describe studies in greater detail than DDI-Codebook.

The DDI-RDF vocabulary is developed with this in mind - discovery should be the same regardless of which DDI-XML specification was used to document a study. The core metadata for the discovery purpose is available in DDI-Codebook and DDI-Lifecycle, which is the reason why the transformation is automated and standardized for both. This means that regardless of the input the resulting RDF is the same which (1) enables an easy and equal search in RDF resulting from DDI-Codebook and DDI-Lifecycle and (2) increases the interoperability between both.

The goal of making this implementation is to provide a simple way to start publishing DDI-XML as RDF. XSLT is also easy to customize and extend, so users can take the base and add output to other vocabularies if they have specialized requirements. It can also be adjusted if special requirements to the input are given. Keeping the XSLT as general as possible, we provide the basis for a broad reusability of the conversion process.

The mappings, including XPath expressions and links to relevant documentation for DDI-Codebook and DDI-Lifecycle, are themselves represented in RDF (see Figure 3.18) in order to be able to execute SPARQL queries on them. Bidirectional mappings of terms from other vocabularies, which are reused multiple times in DDI-RDF, need a context to guarantee unambiguous mapping paths: skos:notation, e.g., is used to represent variable labels and numeric codes of variable code lists. Context information can either be stated in form of a SPARQL query or an informal textual description.

The *Microdata Information System (MISSY)* [28–31, 170] is an online service platform that provides systematically structured metadata for official

[38] Online available at: https://github.com/linked-statistics/DDI-RDF-tools

Fig. 3.18. Bidirectional Mappings between DDI-RDF and DDI-XML

statistics including data documentation at the study and variable level[39] as well as documentation materials, tools, and further information. We developed (1) an editor[40] in compliance with DDI-Codebook, DDI-Lifecycle, and DDI-RDF to improve and simplify the process of documentation and (2) a web information system[41] to provide the end user with various views on the metadata [324].

We use DDI-RDF as core data model and extend it as DDI-RDF does not meet all project requirements. We provide open-source reference implementations of the DDI-RDF and the project-specific data model in Java.[42] As instances of these data models may be physically stored in multiple formats such as DDI-XML, DDI-RDF, relational databases, and Java, we offer implementations for each persistence strategy according to individual persistence APIs. Diverse export routines (e.g., DDI-RDF and DDI-Lifecycle) are available enabling the reuse of metadata in other systems.

3.4 Conclusion

As there are different types of research data and related metadata, we give an overview and explain which vocabularies are commonly used to represent them in RDF. We (1) define each of these types as stated by data professionals, (2) delineate their transition in the research data lifecycle, and (3) depict how metadata on each type of research data is represented using XML standards before respective vocabularies have been developed.

Because of the lack of vocabularies, just a few types of research data and related metadata can be expressed in RDF, which is the reason why we have developed three missing vocabularies (1) to represent all types of research data and its metadata in RDF and (2) to validate RDF data according to constraints extractable from these vocabularies:

- The *DDI-RDF Discovery Vocabulary (DDI-RDF)* [42] supports the discovery of metadata on *unit-record data*, the type of data most often used

[39] 6 series, 74 studies, 129 data sets, 31,834 variables, and 7,922 questions
[40] http://www.gesis.org/missy/editor
[41] http://www.gesis.org/missy
[42] DDI-RDF reference implementation: https://github.com/missy-project

in research within the SBE sciences, i.e., data collected about individuals, businesses, and households. It can be applied to research data from many different domains, rather than being specific to a single set of domain data. DDI-RDF is based on a metadata standard, composed of almost twelve hundred metadata fields, known as the *Data Documentation Initiative (DDI)* [95], an XML format to disseminate, manage, and reuse data collected and archived for research.

- *Physical Data Description (PHDD)* [321] is a vocabulary to describe data in tabular format and its physical properties. The data could either be represented in form of records with character-separated values (CSV) or with fixed length.
- The *SKOS Extension for Statistics (XKOS)* [84] is a vocabulary to describe the structure and textual properties of formal statistical classifications as well as relations between classifications and concepts, and to introduce refinements of SKOS semantic properties to allow the use of more specific relations between concepts.

Acknowledgements

Linked Data community members and statistical domain experts, i.e., representatives of national statistical institutes and national data archives as well as core members of the DDI Alliance Technical Committee - 26 persons from 23 organizations and 12 countries - have developed three RDF vocabularies: the *DDI-RDF Discovery Vocabulary (DDI-RDF)*, *Physical Data Description (PHDD)*, and the *SKOS Extension for Statistics (XKOS)*.

As members of the *RDF Vocabularies Working Group*, an international working group hosted by the international standards organization *DDI Alliance*, we have taken key responsibilities when developing these vocabularies. As editors of the DDI-RDF and PHDD specifications, we are in charge of (1) creating the official HTML specifications and formal specifications in OWL, (2) collecting and collaboratively solving open issues, (3) organizing the technical and the public review phases, and (4) the publication process itself.

The work on these three vocabularies has been started at the first *Dagstuhl Workshop on Semantic Statistics*[43] at Schloss Dagstuhl, Leibniz Center for Informatics, in 2011, and has been continued at the follow-up *Workshop on Semantic Statistics*[44] in the course of the European DDI User Conference, the second *Dagstuhl Workshop on Semantic Statistics*[45] in 2012, and another workshop in 2013. When taking an active part in these workshops, we have been responsible to (1) adequately design the conceptual models for these vocabularies in close cooperation with domain experts and in form of UML

[43] http://www.dagstuhl.de/en/program/calendar/evhp/?semnr=11372

[44] http://www.iza.org/conference_files/EDDI2011/call_for_papers/EDDI11_Program_2011-11-21.pdf

[45] http://www.dagstuhl.de/de/programm/kalender/evhp/?semnr=12422

class diagrams and additional textual descriptions, (2) take, reason, and document design decisions on the model level, and (3) formalize the conceptual models in RDFS/OWL in a semantically correct way. To present DDI-RDF and PHDD to the general public, we held two tutorials at the *4th and the 6th European DDI User Conference*[46] and got another tutorial accepted to be hold at the *10th International Conference on Semantic Computing*.[47]

Chapter 3 is based on 13 publications (4 journal articles, 2 articles in conference proceedings, 2 articles in workshop proceedings, 2 specifications, and 3 technical reports). In [40], we provide a summary of all types of research data and associated metadata and demonstrate the benefits having the DDI-RDF vocabulary for the Linked Data community, data professionals, and the SBE sciences. In [50, 53, 54, 318], we discuss use cases related to DDI-RDF. In [41], we describe the conceptual model of DDI-RDF in a comprehensive way, and in [56], we demonstrate which other vocabularies are reused within the conceptual model of DDI-RDF as well as relationships to other vocabularies. In [40, 41], we give an overview of DDI-XML, the underlying XML standard of DDI-RDF, and in [52], we describe the relations between DDI-XML and DDI-RDF and how to transform DDI-XML documents corresponding to DDI XML Schemas into an RDF representation conforming to DDI-RDF.

At two Dagstuhl workshops,[48] we have pushed the model-driven further development of the DDI standard itself [26]. As members of the *DDI Moving Forward Project*,[49] an international working group of the DDI Alliance, we work with the modeling team to technically formalize the conceptual model of the DDI standard by means of UML 2. As the model-driven development enables to break the model down to diverse concrete bindings such as RDF-S/OWL, XML Schema, relational database schemas, and Java libraries, we contribute to the definition and implementation of diverse model serializations, first and foremost RDFS and OWL.

We have published a journal article to present an overview of several representative applications that use Semantic Web technologies and highlight applications in the SBE sciences such as the Microdata Information System (MISSY)[50] whose data model is based on DDI-RDF [55]. Another journal article serves to show how to enrich social science study descriptions with various data sets from the LOD cloud, expose selected elements of study descriptions in RDF by applying commonly used RDF vocabularies like DDI-RDF, and link study descriptions to adequate entities of external data sets [277].

---

[46] www.eddi-conferences.eu/ocs/index.php/eddi/eddi12/schedConf/program www.eddi-conferences.eu/ocs/index.php/eddi/eddi14/schedConf/program

[47] http://www.ieee-icsc.com

[48] *DDI Lifecycle: Moving Forward*: http://www.dagstuhl.de/de/programm/kalender/evhp/?semnr=12432; *DDI Lifecycle: Moving Forward (Part 2)*: http://www.dagstuhl.de/de/programm/kalender/evhp/?semnr=13442

[49] https://ddi-alliance.atlassian.net/wiki/pages/viewpage.action?pageId=491703

[50] http://www.gesis.org/missy

4

RDFication of XML Enabling to use RDF Validation Technologies

XML is commonly used to (1) represent a large set of information, (2) exchange data in distributed environments, and (3) integrate data from various sources and domains. In the last decade, the *Extensible Markup Language (XML)* [60] has reached wide acceptance as the de facto standard for data exchange as (1) XML is both human readable and machine interpretable, (2) XML is simple to use, and (3) there is a clean separation between the conceptual level with XML Schemas and the instance level with XML documents. An XML document may be an instance of an *XML Schema (XSD)* [304], the primary, widely adopted, and mostly used language for (1) determining the terminology of particular domains, (2) defining structural constraints on sets of XML documents, and (3) validating XML documents on these constraints.

The *Resource Description Framework (RDF)* [89, 143] is the standard data format of the Semantic Web. RDF data is published in the increasingly popular and widely adopted LOD cloud[1] [278] to get linked with a huge number of RDF data sets of different topical domains.[2] As RDF is an established standard, there is a plethora of tools which can be used to interoperate with data represented in RDF which may semantically conform to OWL ontologies.

The *Web Ontology Language (OWL)* [27] is an expressive language used to formally specify the semantics of conceptual models about data and therefore enables software to understand and properly process data according to the intended semantics. OWL has become a popular standard for data representation, data exchange, and data integration of heterogeneous data sources. In combination with the OWL-based *Semantic Web Rule Language (SWRL)* [154], OWL provides facilities for developing very powerful reasoning services enabling to derive implicit knowledge out of explicitly stated knowledge.

OWL ontologies and XSDs are both extensively used to describe conceptual models about data in various domains. There is already a huge amount of XSDs describing conceptual models of many domains, but compared to XSDs

[1] http://lod-cloud.net
[2] http://linkeddata.org

there are only a few ontologies formally representing the intended semantics of particular domains. This is the reason why the LOD cloud is still missing conceptual descriptions [169]. In the B2B domain, e.g., there are hundreds of XSDs to encode exchanged XML data, but not many ontologies.

As syntactic structures of XML documents may be quite complex, they are not intended to be used for information retrieval tasks. Translating XML into RDF, however, enables to formulate queries like *Who is the author of the book 'A Brief History of Time'?* in a semantic way using intuitive terms of respective domains such as *Book* and *author*, instead of formulating queries syntactically on XML documents according to their syntactic structures using rather complex and hard to understand XPath [270] or XQuery [271] expressions.

Traditionally, domain experts work in close collaboration with ontology engineers to design OWL domain ontologies from scratch by hand which requires a lot of time and effort. In many cases, however, XSDs adequately representing particular domains of interest have already been produced by the XML community and all the information located in XSDs can therefore be reused as a basis to develop more sophisticated domain ontologies. Time-consuming work has already been done and domain experts do not have to define the domain data model completely anew. Saved time and effort could be used more effectively to enrich the knowledge representation of given domains with additional domain-specific semantic information not or not satisfyingly covered by already existing XSDs.

Data practitioners 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 XSDs adequately representing particular domains, we propose on formal logics and the XSD meta-model based automatic transformations of arbitrary XSDs and conforming XML documents into OWL ontologies and corresponding RDF data. This does not cause any additional manual effort as these constraints have already been defined by the XML community within underlying XSDs.

Contribution 2-1 *Direct validation of XML data using common RDF validation tools against semantically rich OWL axioms extracted from XML Schemas adequately representing particular domains*

Semantically rich OWL axioms against which XML data can directly be validated are (1) sub-class relationships, (2) OWL hasValue restrictions on data properties, and (3) OWL universal restrictions on object properties. Such universal restrictions could ensure that (1) particular XML elements like the *title* of a book only contain plain text, (2) specific XML attributes such as publication *year* only include positive numbers, or (3) an XML element like *book* only contains listed XML elements (e.g., *isbn*, *title*, and *author*) in a

predefined order. The subsequent excerpt of an XML document contains the book *A Game of Thrones* which is valid in compliance with these constraints. When validating the book *The Sign of the Four*, in contrast, constraint violations are raised since (1) the explicit order of the child elements of the element *book* is not adhered, (2) the attribute *year* includes textual content, and (3) the child element *isbn* is absent.

```
XML:
<library>
    <book year="1996">
        <isbn>0-553-10354-7</isbn>
        <title>A Game of Thrones</title>
        <author>
            <name>George R. R. Martin</name>
        </author>
    </book>
    <book year="February 1890">
        <author>
            <name>Arthur Conan Doyle</name>
        </author>
        <title>The Sign of the Four</title>
    </book>
</library>
```

The subsequent automatically extracted universal restrictions[3] can directly be used to validate RDF data conforming to generated ontologies and therefore underlying XSDs to ensure that book *title* elements and book *id* attributes can only have string literals as content (1+2) and that *book* elements only include the child elements *title* and *author* in exactly this order determined by an XSD sequence inside an XSD complex type definition (3+4):

(1) Title $\sqsubseteq \forall$ value . String
(2) ID $\sqsubseteq \forall$ value . String
(3) Book-Sequence $\sqsubseteq \forall$ contains . (Title $\sqcup$ Author)
(4) Book-Sequence $\sqsubseteq \forall$ sequence . (Title $\sqcup$ Author)

According to the third universal restriction, *book* child element sequences can only contain either *title* or *author* elements or both elements. To fully reflect the intended meaning of the strict order of child elements, however, an additional universal restriction is necessary. As RDF and therefore RDF-based OWL is set-oriented, we introduce the object property *sequence*. Consequently, the fourth universal restriction on the object property *sequence* is used to formally specify the explicit strict order of *book* child elements.

After automatically performed transformations, we are able to validate the resulting RDF data against OWL axioms extracted out of XSDs either (1) explicitly according to XSD constructs determining the syntactic structure of sets of XML document instances or (2) implicitly according to the implicit

[3] OWL axioms are formally defined using Descriptions Logics; IRIs are abbreviated for the sake of simplicity; Classes representing XML elements (e.g., *Title*) can be easily transformed into data properties (e.g., *title*) when deriving domain ontologies in the second step of the proposed semi-automatic approach.

semantics of XSD constructs like class memberships of XML elements or relationships between XML elements. To formally define and to model explicit and implicit semantics in a semantically correct way, we formally underpin transformations themselves on semantically rich OWL axioms for which the knowledge representation formalism *Description Logics* [12, 13, 196, 272] with its well-studied theoretical properties provides the foundational basis.

Contribution 2-2 *Transformations are formally based on semantically rich OWL axioms*

On the meta-model level, we map the XSD meta-model to OWL classes and OWL universal restrictions on data and object properties. These classes, data, and object properties are themselves part of an OWL ontology we have developed to represent the XSD meta-model in OWL. On the schema level, we translate XSDs into sub-classes of these classes, OWL hasValue restrictions on these data properties, and OWL universal restrictions on these object properties.

As structures of XSDs may be quite complex, we base these transformations on the XSD meta-model and map each construct of the XSD meta-model to suitable constructs of an OWL TBox. Such meta-model based transformations enable to translate arbitrary complex structures of XSDs and therefore ensure that any XSD can be converted using identical transformation rules.

Contribution 2-3 *Any XML Schema can be converted to an OWL ontology without any information loss*

We achieve completeness when (1) mapping the XSD meta-model to OWL on the meta-model level, (2) converting XSDs to OWL on the model level, and (3) translating XML data on the instance level. Consequently, there is no information loss during the transformation process of XSDs since all information about the terminology of particular domains, the implicit semantics of XSD constructs, and the syntactic structure of sets of XML documents is maintained and modeled with correct semantics in form of suitable OWL axioms.

A main difference between the structural level of XML and the semantic level of RDF is that XML elements are explicitly ordered and RDF is set-oriented. As XSDs are used to specify structural relationships of objects in data-centric XML documents, we preserve all the structural information of XSDs determining the syntactic structure of sets of XML documents.

Contribution 2-4 *Complete extraction of XML Schemas' structural information*

The XSD construct *sequence*, e.g., is used to specify the explicit order of XML elements contained in parent XML elements and the XSD construct

choice specifies an exclusive or of its operands. 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:

```
XML Schema (excerpt):
<xsd:element name="publication">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:choice>
                <xsd:element name="isbn" type="xsd:string"/>
                <xsd:element name="issn" type="xsd:string"/>
            </xsd:choice>
            <xsd:element name="title" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

Invalid XML (excerpt):
<publication>
    <issn>1744-263X</issn>
    <isbn>1744-263X</isbn>
    <title>International Journal of Metadata, Semantics and Ontologies</title>
</publication>
```

We adequately model the complete syntactic relationships of XSD components and fully appreciate how components relate to other ones on all three levels of instance, schema and meta-model.

Generated ontologies are not directly as useful as manually created domain ontologies as they are not conform to the highest quality requirements of more sophisticated domain ontologies regarding the intended semantics of given domains, since XSDs only transport information about (1) the terminology of individual domains, (2) the syntactic structure of sets of XML documents, and (3) implicit semantics of XSD constructs with limited capabilities. By automatically deriving domain ontologies out of these generated ontologies using manually defined SWRL rules, however, we (1) reduce the complexity of the generated ontologies and thus the underlying XSDs and (2) further supplement OWL axioms with additional domain-specific semantic information not or not satisfyingly covered by underlying XSDs.

The following minimal example shows how the complexity of simplest structures of an XSD describing books and conforming XML is reduced by deriving an excerpt of an OWL domain ontology and conforming RDF data:

```
XML Schema:
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="library">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="book" maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="title" type="xsd:string"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
```

```
        </xsd:complexType>
    </xsd:element>
</xsd:schema>

RDFS/OWL:
:Book a owl:Class .
:title a owl:DatatypeProperty ;
    rdfs:domain :Book ;
    rdfs:range xsd:string .

XML:
<library>
    <book>
        <title>The Hound of the Baskervilles</title>
    </book>
</library>

RDF:
:The-Hound-Of-The-Baskervilles a :Book ;
    :title "The Hound of the Baskervilles" .
```

By evaluating the proposed approach, we verify the hypothesis that the effort and the time needed to deliver high quality domain ontologies from scratch by reusing information of already existing XSDs properly describing particular domains is much less than creating domain ontologies completely manually and from the ground up. The resulting domain ontologies are as usable as ontologies that are entirely constructed by hand, but with a fraction of necessary effort.

Contribution 2-5 *Minimized effort and time designing OWL domain ontologies from scratch when XML Schemas, adequately describing conceptual models about data of particular domains, have already been developed*

Chapter Overview

Section 4.1 serves to provide an overview of the individual stages of the overall process semi-automatically deriving OWL domain ontologies based on already existing XSDs adequately describing particular domains. In Section 4.2, we depict how we mapped each component of the XSD meta-model to suitable OWL constructs of an ontology representing the XSD meta-model in OWL. The conceptual mappings for the transformations of XSDs into OWL on the schema level, which are based on the proposed XSD meta-model ontology, are delineated in Section 4.3.

To get a better idea of (1) how XSDs and XML documents are translated into OWL ontologies' TBoxes and conforming ABoxes and (2) how domain ontologies are derived out of generated ontologies on the schema and the instance level, we present the application of the proposed approach in form of a complete case study and this way motivate its usage (see Section 4.4). By means of an intuitive running example, we demonstrate a fast way to disseminate the huge amount of already existing XSDs and XML documents of the commonly accepted and widely used XML standards DDI-Codebook and

DDI-Lifecycle. We show how we make profit of all the work which has already been done by the DDI community and derive the OWL domain ontology DDI-RDF.

In Section 4.5, we accurately describe the implementation of the automatic transformations of XSDs into OWL ontologies. The evolved concept behind the implementation enables to manage just with XML technologies as the mapping itself is completely realized using XSLT.

In Section 4.6, we provide a comprehensive evaluation of our approach. The first step is to automatically transform XSDs into OWL ontologies. To prove the generality of these transformations, we show that any XSD can be converted to OWL by executing generic test cases created out of the XSD meta-model. In addition, we converted XSDs of six widely known, accepted, and used XML standards from the academic (i.a., DDI-Lifecycle) and industrial field. The second step is to define SWRL rules on the schema level by hand to derive OWL domain ontologies automatically out of the generated ontologies on both the instance and schema level. We specified SWRL rules for three domain ontologies - two from the industrial and one from the academic area. To verify the hypothesis, we (1) determined the effort and expenses for the traditional manual approach and (2) estimated the effort and expenses for the suggested semi-automatic approach. DDI-RDF serves as domain ontology, since we were part of the process creating it manually from scratch.

4.1 Designing OWL Domain Ontologies based on XML Schemas

In this section, we give an overview of the individual stages of the overall process semi-automatically deriving OWL domain ontologies based on already existing XSDs adequately representing particular domains.

Mapping of the XML Schema Meta-Model to OWL

XML documents may be instances of XSDs determining the terminology of certain domains and the syntactic structure of sets of XML documents. XSDs themselves are instances of the XSD meta-model, the XML Schema for XML Schemas, or also called the *XSD abstract data model* [304]. The W3C defined XSD, the class of XML documents, recursively using the XSD language to describe the XSD language, just like XSD documents are XML documents describing XML documents.

On the meta-model level M2, we mapped all components of the XSD meta-model to classes, universal restrictions on data properties, and universal restrictions on object properties of the *XSD Meta-Model Ontology*, an ontology we defined to represent the XSD meta-model in OWL (see Section 4.2).

Multi-Level Semi-Automatic Process Designing Domain Ontologies Based on XML Schemas

The multi-level process designing domain ontologies based on already available XSDs [46] is divided into the following consecutive steps (see Figure 4.1):

1. XSDs and XML documents are automatically transformed into OWL ontologies and conforming RDF data (see Section 4.3)
   1.1. XSDs are converted to OWL ontologies on level M1 using XSLT [176]
   1.2. XML documents are translated into RDF data corresponding to these OWL ontologies on level M0 using Java
2. OWL domain ontologies and conforming RDF data are automatically derived by means of manually defined SWRL rules (see Section 4.4)
   2.1. SWRL rules are manually specified on level M1
   2.2. Reasoning is automatically executed on the generated ontologies on levels M0 and M1 using the predefined SWRL rules

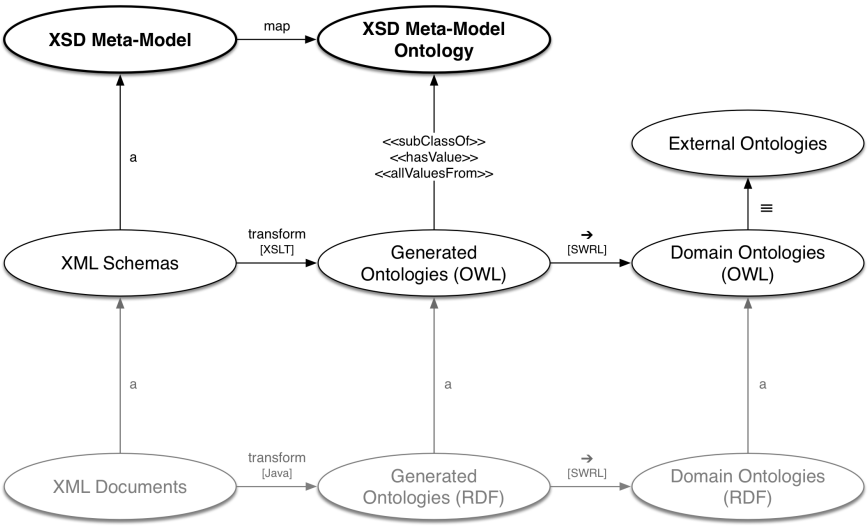

Fig. 4.1. Multi-Level Process Designing Domain Ontologies Based on XML Schemas

Automatic Transformations of XML Schemas and XML Documents into OWL Ontologies and RDF Data

The knowledge representation formalism *Description Logics* [12, 13, 196] with its well-studied theoretical properties provides the foundational basis for the transformations of XSDs and conforming XML documents into OWL ontologies and corresponding RDF data. On the schema level M1, we automatically

convert XSDs to (1) OWL sub-classes of the XSD Meta-Model Ontology's classes, (2) OWL hasValue restrictions on data properties of the XSD Meta-Model Ontology, and (3) OWL universal restrictions on object properties of the XSD Meta-Model Ontology using XSLT transformations. As each component of the XSD abstract data model is covered, any XSD can be translated into an OWL ontology. On the instance level M0, XML documents are converted to ABoxes of generated ontologies using Java. We use Java and not XSLT as Java scales and performs better when transforming very large XML documents.

Semi-Automatic Derivation of OWL Domain Ontologies

We use *SWRL rules* [154] (1) to automatically deduce domain ontologies on the schema level M1 and (2) to automatically infer RDF data corresponding to domain ontologies on level M0 out of generated ontologies and conforming RDF data. Ontology engineers and domain experts may work together to define SWRL rules manually only on the schema level M1. SWRL rules are executed by rule engines like *Pellet* [285], the OWL 2 reasoner for Java. The *antecedents* of SWRL rules are specified according to syntactic structures of XML documents. The *consequents* of SWRL rules, on the other side, are defined according to conceptual models of domain ontologies. Consequently, conceptual models have to be devised in a first stage. Finally, XML data conforming to XSDs can be converted to instances of domain ontologies.

Domain ontologies' classes and properties can be annotated as being equivalent to classes and properties of commonly accepted and widely adopted ontologies such as SKOS [219] and the *DCMI Metadata Terms* [103]. As a consequence, reasoners may use additional semantics defined by other ontologies for their deductions [197].

4.2 Mapping of the XML Schema Meta-Model to OWL

When representing the components of XSDs in XML, they are called *element information items (EIIs)*. On the meta-model level M2, we mapped all meta EIIs to classes, universal restrictions on data properties, and universal restrictions on object properties of the *XSD Meta-Model Ontology*, an OWL ontology we defined to represent the XSD meta-model in OWL. Tables 4.1 and 4.2 sketch these mappings. We use Description Logics to formally represent OWL language constructs.

Meta-EIIs

We mapped meta-EIIs to classes of the XSD Meta-Model Ontology. The class *Element*, for instance, stands for the XSD meta-model's meta-EII *element*.

Table 4.1. Mapping of the XML Schema Meta-Model to OWL (1)

XSD meta-model	XSD Meta-Model Ontology
meta-EIIs	classes: `<meta-EII>`
attributes of meta-EIIs	1. data properties: `<attribute>_<domain meta-EII>_String` 2. universal restrictions on data properties: `<domain meta-EII>` $\sqsubseteq$ $\forall$ `<attribute>_<domain meta-EII>_String.String`
any well-formed XML content of meta-EIIs Appinfo | Documentation	1. data properties: `any_<Appinfo|Documentation>_String` 2. universal restrictions on data properties: `<Appinfo|Documentation>` $\sqsubseteq$ $\forall$ `any_<Appinfo|Documentation>_String.String`
texts contained in elements and attributes of XML documents	1. data properties: `value_<Element|Attribute>_String` 2. universal restrictions on data properties: `<Element|Attribute>` $\sqsubseteq$ $\forall$ `value_<Element|Attribute>_String.String`
attributes of meta-EIIs referring to meta-EIIs (attributes *ref*, *refer substitutionGroup*)	1. object properties: `<ref|substitutionGroup|refer>_<domain meta-EII>_range meta-EII>` 2. universal restrictions on object properties: `<domain meta-EII>` $\sqsubseteq$ $\forall$ `<ref|substitutionGroup|refer>_<domain meta-EII>_range meta-EII>.<range meta-EII>`

Attributes of Meta-EIIs

Attributes of meta-EIIs are mapped to

1. data properties `<attribute>_<domain meta-EII>_String` with the class standing for the domain meta-EII as domain and the class representing the XSD built-in primitive datatype *xsd:string* as range, and
2. universal restrictions on these data properties `<domain meta-EII>` $\sqsubseteq \forall$ `<attribute>_<domain meta-EII>_String.String` which express that the class including all domain meta-EII individuals is defined as the sub-class of the anonymous complex super-class of all the instances which can only have relationships along these data properties to literals of the datatype *xsd:string* or have no relationships via these data properties.

The attribute *name* of the meta-EII *element*, e.g., is mapped to the data property `name_Element_String` and to the data property's universal restriction `Element` $\sqsubseteq \forall$ `name_Element_String.String` as elements can only have `name_Element_String` relationships to *String* literals.

Any Well-Formed XML Content of Meta-EIIs *Appinfo* or *Documentation*

The meta-EIIs *Appinfo* and *Documentation* may comprise any well-formed XML content such as XML elements, XML attributes, and plain text. For this reason, any well-formed XML content of the meta-EIIs *Appinfo* and *Documentation* is mapped (1) to the data properties `any_<Appinfo|Documentation>_String` and (2) to the universal restrictions on these data properties `<Appinfo|Documentation> ⊑ ∀ any_<Appinfo|Documentation>_String.String`.

Texts Contained in Elements and Attributes of XML Documents

Elements and attributes of XML documents may comprise text. Thus, we added (1) the data properties `value_<Element|Attribute>_String` and (2) universal restrictions on these data properties `<Element|Attribute> ⊑ ∀ value_<Element|Attribute>_String.String` to the XSD Meta-Model Ontology. On the instance level, the XML document excerpt `<Label lang="en">Age</Label>` is converted to the property assertions `value_Element_String (Label-Individual ..., 'Age')` and `value_Attribute_String (Lang-Individual...,'en')`.

Attributes of Meta-EIIs Referring to Meta-EIIs (Attributes *ref*, *refer*, and *substitutionGroup*)

Attributes of meta-EIIs like *ref*, *refer*, and *substitutionGroup* referring to other meta-EIIs are mapped (1) to the object properties `<ref|substitutionGroup|refer>_<domain meta-EII>_<range meta-EII>` and (2) to the universal restrictions on these object properties `<domain meta-EII> ⊑ ∀ <ref|substitutionGroup|refer>_<domain meta-EII>_<range meta-EII>.<range meta-EII>`. According to the object property's universal restriction `Element ⊑ ∀ ref_Element_Element.Element`, e.g., elements can only have `ref_Element_Element` relationships to elements or no such relations.

Attributes of Meta-EIIs Referring to Type Definitions (Attributes *type* and *base*)

Meta-EIIs' attributes *base* and *type* refer to simple ur-type, simple type, or complex type definitions.

An *ur-type definition* [24] is present in each XSD and serves as the root of the type definition hierarchy for that schema. A *simple type definition* [24] is a set of constraints which applies to the values of attributes and the text-only content of elements. Each simple type definition, whether a built-in primitive XSD datatype or user-defined, is a restriction of some particular simple base type definition, the *simple ur-type* [24].

A *complex type definition* [24] is a set of attribute declarations and a content type, applicable to the attributes and children of an EII respectively. The

Table 4.2. Mapping of the XML Schema Meta-Model to OWL (2)

XSD meta-model	XSD Meta-Model Ontology
attributes of meta-EIIs referring to type definitions (attributes *type* and *base*)	1. object properties: `<type|base>_<domain meta-EII>_Type` 2. universal restrictions on object properties: `<domain meta-EII>` ⊑ ∀ `<type|base>_<domain meta-EII>_Type.Type`

attribute *memberTypes*	1. object property:
	`memberTypes_union_Type`
	2. universal restriction on the object property:
	`<union>` ⊑ ∀ `memberTypes_union_Type.Type`

meta-EIIs' part-of relationships	1. object properties:
	`contains_<domain meta-EII>_<range meta-EII>`
	2. universal restrictions on object properties:
	`<domain meta-EII>` ⊑
	∀ `contains_<domain meta-EII>_<range meta-EII>.`
	`<range meta-EII>`

sequence of in meta-EII *sequence* contained meta-EIIs	1. object property:
	`sequence`
	2. universal restrictions on the object property:
	`<sequence>` ⊑ ∀ `sequence.<range meta-EII>`

content type may require the children to contain neither EIIs nor character information items (that is, to be empty), to be a string which belongs to a particular simple type or to contain a sequence of EIIs which conforms to a particular model group, with or without character information items as well.

1. Following the same applied mapping procedure, these attributes would be mapped to six object properties `<type|base>_<domain meta-EII>_SimpleType|AnySimpleType|ComplexType`. XSLT transformations generating OWL ontologies out of XSDs would have to determine the object properties' range classes *AnySimpleType*, *SimpleType*, and *ComplexType* as part of the object properties' identifiers at runtime. If the attributes *type* or *base* either point to simple or complex type definitions defined in external XSDs, these XSDs would have to be physically available to traverse their XML trees and to iterate over each simple and complex type definition. In many cases, however, external XSDs are not physically available. For this reason, we mapped the attributes *type* and *base* to the object properties `<type|base>_<domain meta-EII>_Type` with the range class *Type* representing the super-class of all three specific type definitions.
2. The attributes *base* and *type* are also mapped to the object properties' universal restrictions `<domain meta-EII>` ⊑ ∀ `<type|base>_<domain meta-EII>_Type.Type`. Considering the object property's universal restriction `Element` ⊑ ∀ `type_Element_Type.Type`, elements can only have `type_`

`Element_Type` relationships to *Type* individuals, which are simple or complex type definitions in this case, or have no such relations.

Attribute *memberTypes*

On the schema level M1, the attribute *memberTypes* of the EII *union* may include simple ur-type and simple type definitions separated by blank characters. Thus, the attribute *memberTypes* is mapped (1) to the object property `memberTypes_union_Type` and (2) to the object property's universal restriction `<union>` $\sqsubseteq \forall$ `memberTypes_union_Type.Type`.

Meta-EIIs' Part-Of Relationships

Meta-EIIs may contain other meta-EIIs. For this reason, (1) the object properties `contains_<domain meta-EII>_<range meta-EII>` and (2) associated universal restrictions `<domain meta-EII>` $\sqsubseteq \forall$ `contains_<domain meta-EII>_<range meta-EII>.<range meta-EII>` are specified. In accordance with the object property's universal restriction `Sequence` $\sqsubseteq \forall$ `contains_Sequence_Element.Element`, sequences can only include elements along the object property `contains_Sequence_Element` and no instances of other classes.

Sequence of in Meta-EII 'sequence' Contained Meta-EIIs

The universal restrictions on the object properties `contains_Sequence_<range meta-EII>` state for each range meta-EII *annotation*, *element*, *group*, *choice*, and *sequence*) that range instances have to be of the classes representing these range *meta-EIIs*. The object property *sequence* and the object property's universal restriction `<sequence>` $\sqsubseteq \forall$ `sequence.<range meta-EII>` are added to the XSD Meta-Model Ontology which enables to capture the strict order of the in the EII *sequence* contained EIIs when XSDs are converted to OWL ontologies on the schema level M1.

4.3 Transformations of XML Schemas into OWL Ontologies

On the schema level M1, XSDs are translated into (1) OWL sub-classes of XSD Meta-Model Ontology's classes, (2) OWL hasValue restrictions on XSD Meta-Model Ontology's data properties, and (3) OWL universal restrictions on XSD Meta-Model Ontology's object properties. Like heavyweight ontologies [294], generated ontologies consist of a hierarchy of classes as well as relations with domains and ranges. Tables 4.3 and 4.4 demonstrate these transformations from XSDs into OWL ontologies.

Table 4.3. Transformation of XML Schemas into OWL Ontologies (1)

XSDs	**OWL ontologies**
EIIs	sub-classes of XSD Meta-Model Ontology's classes: <EII> ⊑ <meta-EII>
values of EIIs' attributes	hasValue restrictions on XSD Meta-Model Ontology's data properties: <domain EII> ⊑ ∃ <attribute>_<domain meta-EII>_String.{<String>}
any well-formed XML content of EIIs Appinfo|Documentation	hasValue restrictions on XSD Meta-Model Ontology's data properties: <Appinfo|Documentation> ⊑ ∃ any_<Appinfo|Documentation>_String.{<String>}
values of EIIs' attributes referring to EIIs (attributes *ref*, *substitutionGroup*, *refer*)	universal restrictions on XSD Meta-Model Ontology's object properties: <domain EII> ⊑ ∀ <ref|substitutionGroup|refer>_ <domain meta-EII>_<range meta-EII>.<range EII>

EIIs

EIIs are transformed into sub-classes of the XSD Meta-Model Ontology's classes representing meta-EIIs: <EII> ⊑ <meta-EII>. Hence, all generated OWL ontologies are based on the same reusable OWL classes. To show an example, the XSD's EII *element* with the name *Label* (<xs:element name="Label"/>) is converted into the class *Label-Element...* (the automatically generated unique IRI is obviously much more complex than the stated one) which is defined as sub-class of the class *Element* (Label-Element... ⊑ Element), since on the instance level M0 all *Label* individuals are also part of the *Element* class extension which means that *Label* elements are elements.

Values of EIIs' Attributes

Values of EIIs' attributes are translated into hasValue restrictions on XSD Meta-Model Ontology's data properties <domain EII> ⊑ ∃ <attribute>_<domain meta-EII>_String.{<String>}, as classes representing domain EIIs are defined as sub-classes of the anonymous complex super-classes of all the individuals which have at least one relationship along the data properties <attribute>_<domain meta-EII>_String to the specified individuals of the XSD's primitive datatype *string*. The value of the attribute *name* of the EII *element* (<xs:element name="Label"/>) is transformed into the data property hasValue restriction Label-Element... ⊑ ∃ name_Element_String.{'Label'}, as on the instance level M0 *Label* elements must have a name which is *Label* and which is of the datatype *string*.

Any Well-Formed XML Content of EIIs Appinfo or Documentation

Any well-formed XML content of the EIIs *Appinfo* and *Documentation* such as XML elements, XML attributes, and plain text is converted to hasValue restrictions on XSD Meta-Model Ontology's data properties `<Appinfo|Documentation>` $\sqsubseteq \exists$ `any_<Appinfo|Documentation>_String.{<String>}`. The text contained in the EII *appinfo* (`<xs:appinfo> This is an application information. </xs:appinfo>`) is converted to the data property hasValue restriction `Appinfo1...` $\sqsubseteq \exists$ `any_Appinfo_String.{'This is an application information.'}`.

Values of EIIs' Attributes Referring to EIIs (Attributes *ref*, *refer*, and *substitutionGroup*)

Values of EIIs' attributes *ref*, *substitutionGroup*, and *refer* referring to other EIIs are translated into XSD Meta-Model Ontology's object properties' universal restrictions `<domain EII>` $\sqsubseteq \forall$ `<ref|substitutionGroup|refer>_<domain meta-EII>_<range meta-EII>.<range EII>`. The reference to the global element *Label* (`<xs:element ref="Label"/>`), e.g., is converted to the object property's universal restriction `Label-Element-Reference1...` $\sqsubseteq \forall$ `ref_Element_Element.Label-Element...` .

Table 4.4. Transformation of XML Schemas into OWL Ontologies (2)

XSDs	OWL ontologies
values of EIIs' attributes referring to type definitions (attributes *type* and *base*)	universal restrictions on XSD Meta-Model Ontology's object properties: `<domain EII>` $\sqsubseteq$ $\forall$ `<type|base>_<domain meta-EII>_Type.<range EII>`
values of attribute *memberTypes*	univeral restriction on XSD Meta-Model Ontology's object property: `<union>` $\sqsubseteq$ $\forall$ `memberTypes_Union_Type.<union of Type EIIs>`
EIIs' part-of relationships	universal restrictions on XSD Meta-Model Ontology's object properties: `<domain EII>` $\sqsubseteq$ $\forall$ `contains_<domain meta-EII>_ <range meta-EII>.<union of range EIIs>`
sequence of in EII *sequence* contained EIIs	universal restrictions on XSD Meta-Model Ontology's object property: `<sequence>` $\sqsubseteq \forall$ `sequence.<union of EIIs>`

Values of EIIs' Attributes Referring to Type Definitions (Attributes *type* and *base*)

Values of EIIs' attributes *type* and *base* referring to simple ur-type, simple type, or complex type definitions are converted to universal restric-

tions on XSD Meta-Model Ontology's object properties `<domain EII>` ⊑ ∀ `<type|base>_<domain meta-EII>_Type.<range EII>`. The value 'VariableType' of the attribute *type* of the EII *element* with the name *Variable* (`<xs:element name="Variable" type="VariableType" />`), e.g., is transformed into the object property's universal restriction `Variable-Element...` ⊑ ∀ `type_Element_Type. VariableType-Type...` .

Values of the Attribute *memberTypes*

The attribute *memberTypes* of the EII *union* may contain multiple simple ur-type and simple type definitions separated by blank characters. Consequently, values of this attribute are converted to the XSD Meta-Model Ontology's object property's universal restrictions `<union>` ⊑ ∀ `memberTypes_Union_Type. <union of Type EIIs>`. The attribute *memberTypes*, e.g., contains references to one simple ur-type and two simple type definitions (`<xs:union memberTypes = "SimpleType1 SimpleType2 xs:string"/>`). The value of the attribute *memberTypes* is translated into the object property's universal restriction `Union1...` ⊑ ∀ `memberTypes_Union_Type.(SimpleType1-Type...` ⊔ `SimpleType2-Type...` ⊔ `string-Type...)`.

EIIs' Part-Of Relationships

As EIIs may include one to multiple other EIIs, universal restrictions on XSD Meta-Model Ontology's object properties `<domain EII>` ⊑ ∀ `contains_<domain meta-EII>_<range meta-EII>.<union of range EIIs>` are used to transform EIIs' part-of relationships. The following sequence, e.g., contains only one *element* EII, a reference to the global element *Label*: `<xs:sequence> <xs:element ref="Label"/> </xs:sequence>`. According to the object property's universal restriction `Sequence1 ...` ⊑ ∀ `contains_Sequence_Element.Label-Element-Reference1...`, the range of the object property can only comprise instances of the class representing the reference to the global element *Label*. If EIIs have more than one EII as content, the domain EIIs can only have relationships along particular object properties to individuals of the anonymous complex super-class consisting of the union of multiple classes representing the contained range EIIs. The part-of relationship of the sequence (`<xs:sequence> <xs:element ref="VariableName"/> <xs:element ref="Label"/> </xs:sequence>`), e.g., is transferred into the object property's universal restriction `Sequence1 ...` ⊑ ∀ `contains_Sequence_Element.(VariableName-Element-Reference1...` ⊔ `Label-Element-Reference2...)`.

Sequence of in EII *sequence* Contained EIIs

According to the universal restrictions on the object properties `contains_Sequence_Element|Sequence`, sequences can only have relationships along these object properties to elements or other sequences. Sequences, however, may

not only include either elements or sequences but also annotations, groups, and choices simultaneously. Furthermore, sequences are not only containers of EIIs. They also store the strict order of contained EIIs. As sequences may contain multiple EIIs and in order to store the strict order of in sequences included EIIs, we added the object property *sequence* and the universal restrictions `<sequence>` $\sqsubseteq \forall$ `sequence.<union of EIIs>` to the XSD Meta-Model Ontology. According to the XSD fragment `<xs:sequence> <xs:element ref="VariableName"/><xs:element ref="Label"/></xs:sequence>`, the sequence instance either contains *VariableName* or *Label* individuals representing references to global elements: `Sequence1...` $\sqsubseteq \forall$ `sequence.(VariableName-Element-Reference1...` $\sqcup$ `Label-Element-Reference2...)`. The sequence of EIIs is extracted in compliance with the order of the union operands.

4.4 Case Study

To get a better idea of (1) how XSDs and XML documents are translated into OWL ontologies' TBoxes and ABoxes and (2) how OWL domain ontologies are derived out of generated ontologies on the schema level M1 and the instance level M0, we demonstrate the application of the proposed approach in form of a complete case study and this way motivate its usage. This case study serves to show in detail how excerpts of the OWL domain ontology DDI-RDF are deduced out of XSDs which are part of the DDI XML standard. More specifically, in case specific conditions are fulfilled it is derived that a certain resource represents a social science variable with a particular variable label. In the following, we describe the individual steps of the overall process designing OWL domain ontologies based on already existing XSDs by means of an intuitive running example.

4.4.1 Automatic Transformations of XML Schemas and XML Documents into OWL Ontologies and RDF Data

Figure 4.2 visualizes the input of the transformation process: The XML document (on the right) storing information about variables and the XSD determining the XML document's syntactic structure. XML elements *Variable* may contain XML elements *Label* corresponding to variable labels which may include plain text such as 'Age'. *Variable* is an instance of the XSD EII *element* whose *name* attribute has the value *Variable* and whose *type* attribute has the value *VariableType* referring to the complex type definition *VariableType*. This complex type with the name *VariableType* comprises the EII *complexContent* including the XSD component *extension* which contains a sequence. This sequence comprises a reference to the global element with the name *Label* which is the type of the XML element *Label*. The global XML element *Label* may include strings of the datatype *xsd:string*.

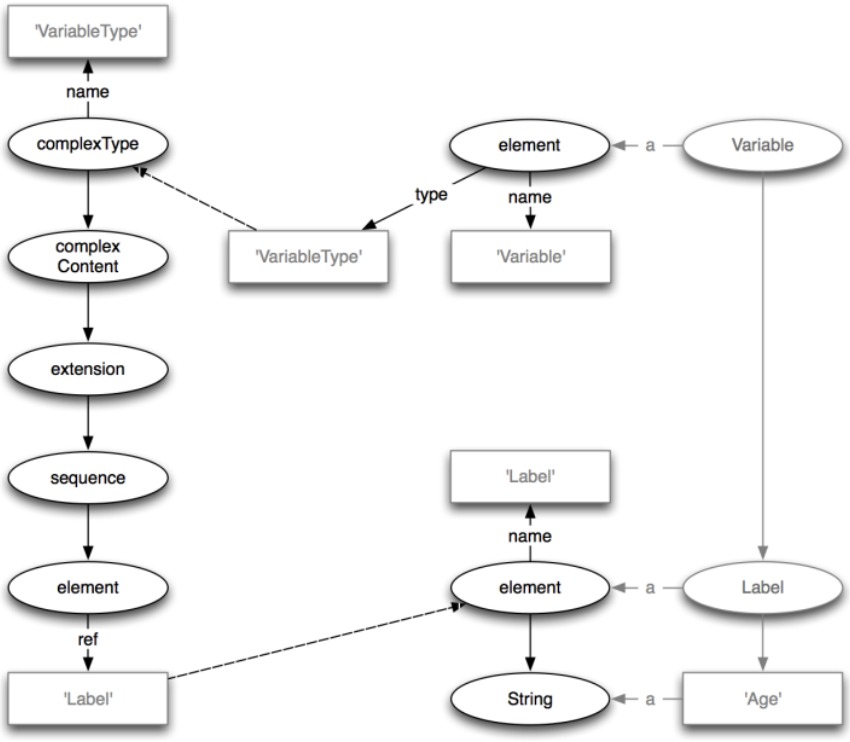

Fig. 4.2. XML Schemas and XML Documents

Overview of the Transformation Process

To be able to use this rich syntactic information for the ontology, instead of just using the instance data, we first transform the schema automatically with generic XSLT transformations to an OWL ontology. The transformation process of XSDs to OWL ontologies on the schema level M1 covering the most important transformation steps is summarized as follows:

1. XSD's EIIs are transformed into OWL classes having globally unique IRIs
2. OWL classes are defined as sub-classes of classes which are specified within the XSD Meta-Model Ontology
3. Values of EIIs' attributes are converted to OWL hasValue restrictions on XSD Meta-Model Ontology's data properties
4. (1) EIIs' part-of relationships, (2) XML elements' and XML attributes' content, (3) EIIs' attributes' values referring to either type definitions or (4) other EIIs, and (5) sequences of in EII *sequence* contained EIIs are transformed into OWL universal restrictions on XSD Meta-Model Ontology's object properties

On the instance level M0, XML documents are converted to RDF corresponding to the generated OWL ontologies. Thereby, XML elements and XML attributes are concrete instances of OWL classes which are generated during the transformation process of XSDs to OWL ontologies. These instances must follow the semantics stated in the generated OWL ontologies and extracted from the underlying XSDs.

1. OWL Classes

The first step is to convert each XSD's EII to an OWL class (see Figure 4.3). The XSLT assigns OWL class identifiers considering the developed naming conventions (see Section 4.5) which ensure the global uniqueness of the IRIs. In contrast to OWL, XSD has very few restrictions on unique naming. IRIs of OWL ontologies have to be quite long to be globally unique. The global element *Variable* (`<xs:element name= "Variable".../>`), e.g., is translated into the class *Variable-Element...* with the meta-EII *element* as part of its identifier.

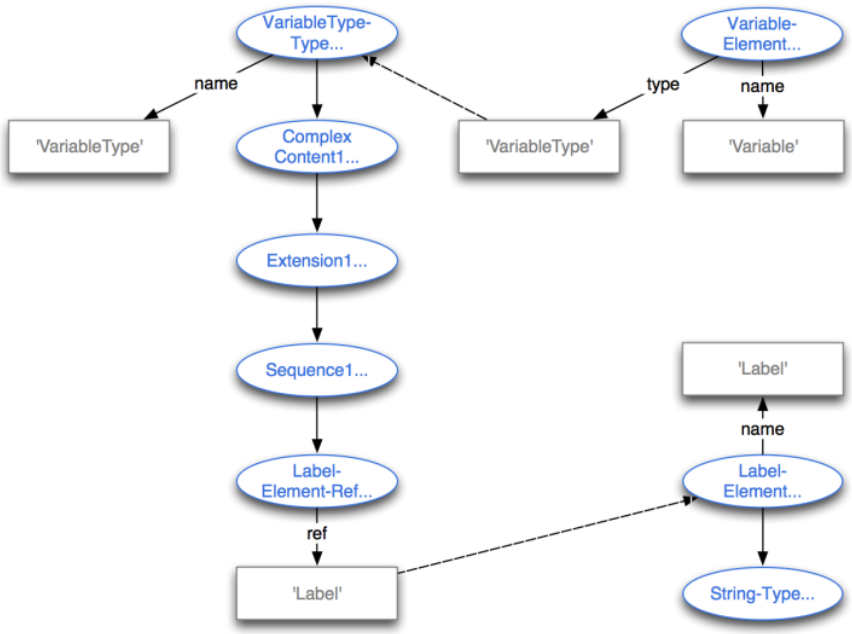

Fig. 4.3. Transformation of XML Schemas' EIIs into OWL classes

2. Sub-Class Relationships

Now, XSD's EIIs are transformed into OWL classes with globally unique IRIs. But so far, the transformation does not cover the XSDs' semantics. Semantics is added by defining sub-class relationships between classes of generated ontologies and classes of the XSD Meta-Model Ontology (see Figure 4.4). OWL classes of generated ontologies are defined as sub-classes of the super-classes specified in the XSD Meta-Model Ontology: `<EII>` $\sqsubseteq$ `<meta-EII>`. Classes standing for specific XSD elements like *Variable-Element...*, e.g., are translated into sub-classes of the super-class *Element* representing the meta-EII *element* (`<Variable-Element...>` $\sqsubseteq$ `<Element>`), as each particular EII *element* is also part of the *Element* class extension, i.e., in more simple terms, each specific XSD element is obviously an XSD element.

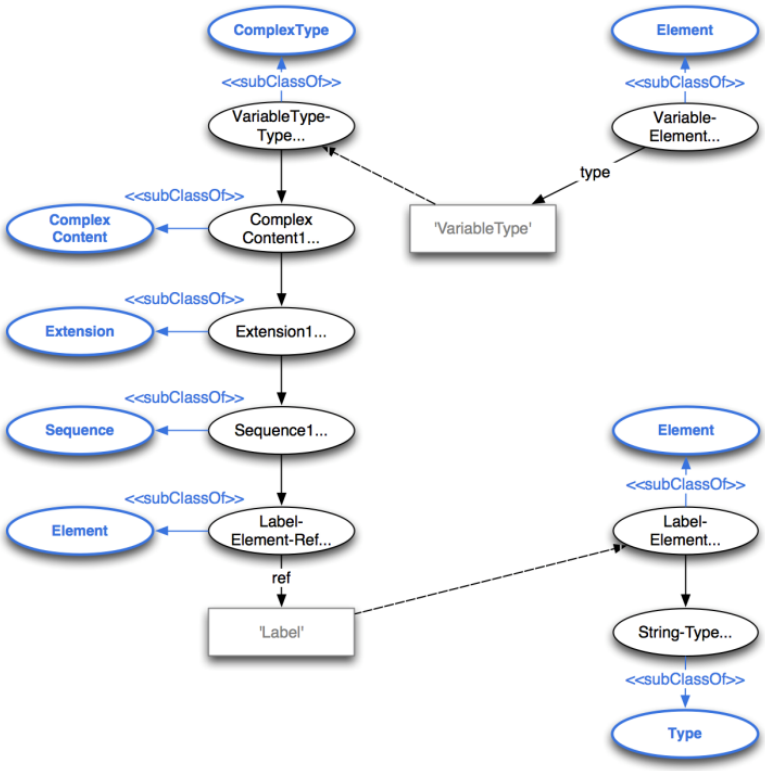

Fig. 4.4. Sub-Class Relationships

3. HasValue Restrictions on Data Properties

So far, the XSD's EIIs are converted to sub-classes of the XSD Meta-Model Ontology's super-classes representing XSD meta-EIIs. As next step EIIs' at-

tributes' values are converted to XSD Meta-Model Ontology's data properties `<attribute>_<domain meta-EII>_String` and to hasValue restrictions on these data properties: `<domain EII>` $\sqsubseteq \exists$ `<attribute>_<domain meta-EII>_String.{<String>}` (see Figure 4.5). The value 'Variable' of the attribute *name* of the EII *element* (`<xs:element name="Variable".../>`), e.g., is translated into the XSD Meta-Model Ontology's data property `name_Element_String` pointing from an *element* to a *string* and into the hasValue restriction on this data property `Variable-Element...` $\sqsubseteq \exists$ `name_Element_String.{'Variable'}`, since *Variable-Element...* resources must have at least one relationship along the data property `name_Element_String` to the string 'Variable'. In other words, variable XSD elements must have the name 'Variable'.

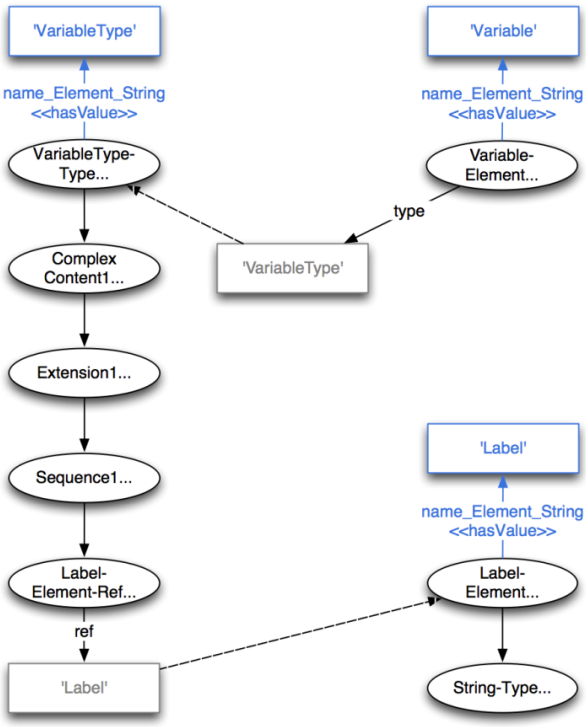

Fig. 4.5. HasValue Restrictions on Data Properties

4. Universal Restrictions on Object Properties

XSD's EIIs and XSD's EIIs' attributes' values are now translated. The last step is to transform (1) EIIs' part-of relationships, (2) XML elements' and attributes' content, (3) EIIs' attributes' values referring to either type definitions or (4) other EIIs, and (5) sequences of in EII *sequence* contained EIIs

into universal restrictions on XSD Meta-Model Ontology's object properties (see Figure 4.6).

Values of EIIs' attributes referring to other EIIs are transformed into XSD Meta-Model Ontology's object properties' universal restrictions `<domain EII>` $\sqsubseteq \forall$ `<ref|substitutionGroup|refer>_<domain meta-EII>_<range meta-EII>.<range EII>`. The value 'Label' of the *element* EII's attribute *ref* (`<xs:element ref="Label"/>`) referring to the EII *element* with the name *Label*, e.g., is translated into the object property `ref_Element_Element` and its universal restriction `Label-Element-Reference1...` $\sqsubseteq \forall$ `ref_Element_Element.Label-Element...`

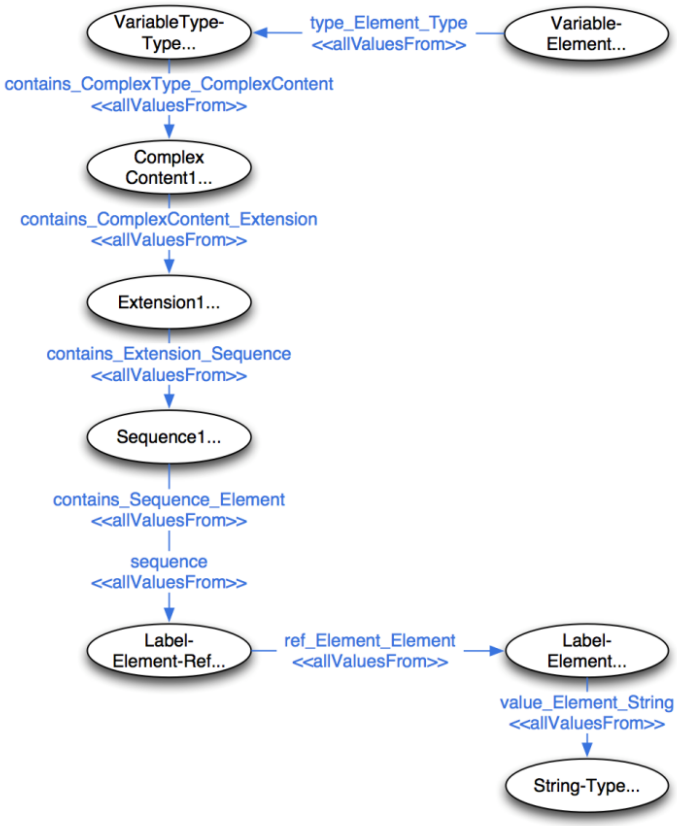

Fig. 4.6. Universal Restrictions on Object Properties

Values of EIIs' attributes referring to type definitions are translated into universal restrictions on XSD Meta-Model Ontology's object properties: `<domain EII>` $\sqsubseteq \forall$ `type|base_<domain meta-EII>_Type.<range EII>`. The value 'VariableType' of the attribute *type* of the EII *element* with the name *Variable* (`<xs:element name="Variable" type="VariableType"/>`), e.g., is con-

verted to the object property's universal restriction `Variable-Element...` ⊑ ∀ `type_Element_Type.VariableType-Type...` .

The part-of relationship of the EII *sequence* is translated into the object property's universal restriction `Sequence1...` ⊑ ∀ `contains_Sequence_Element.Label-Element-Reference1...` . The sequence includes only a reference to the global element *Label*. The strict order of the EIIs contained in the sequence, on the other side, is expressed by the object property's universal restriction `Sequence1...` ⊑ ∀ `sequence.Label-Element-Reference1...` .

As resources of the class *Label-Element...*, i.e., *Label elements* contained in XML documents on the instance level M0, may have text as content, i.e., *String-Type...* literals, the data property `value_Element_String` is introduced and the data property's universal restriction `Label-Element...` ⊑ ∀ `value_Element_String.String` is defined. On level M0, RDF data conforming to the generated OWL ontology may be validated on this universal restriction in order to ensure that the only allowed content of *Label elements* is of the datatype *xsd:string*.

While this means that we have many classes with long names after the transformations, it also means that we adequately model the complete syntactic and semantic relationships of the XSD components. We can fully appreciate how components relate to other ones on all three levels of instance, schema and meta-model. Since this is all automatically generated, this multiplication of information is not detrimental, but instead allows us to use all this data in a way that is fully integrated with each other. At no cost of time for the ontology engineer. Now, all the information located in the underlying XSDs of a specific domain is also expressed in generated OWL ontologies.

4.4.2 Semi-Automatic Derivation of OWL Domain Ontologies

As XSDs' structures and therefore generated ontologies' structures may be quite complex, generated ontologies are not directly as useful as manually created domain ontologies. Therefore, our idea is to perform semi-automatic transformations based on the automatically created ontologies using SWRL rules. By transforming generated ontologies into domain ontologies (1) we reduce the complexity of generated ontologies and (2) we further supplement the OWL axioms of generated ontologies with additional domain-specific semantic information not satisfyingly covered by underlying XSDs. Figure 4.7 visualizes the generated OWL ontology, its RDF representation, the OWL domain ontology's extraction to be derived, and the SWRL rule's atoms.

In our case study, we want to deduce an excerpt of the DDI-RDF vocabulary. More specifically, we want to derive that the resource with the URI *age*, which is assigned to the class *Variable-Element...*, is a *variable* and that the same resource has the variable label 'Age' of the datatype *xsd:string*. Thereby, the data property *skos:prefLabel* represents relationships between *variables* and variable labels. The following code fragment demonstrates the

Fig. 4.7. Derive Domain Ontology

antecedent and the consequent of the SWRL rule which is executed by a rule engine to derive these two statements.

```
1 (?domain type_Element_Type ?a) ∧ (?a contains_ComplexType_ComplexContent ?b) ∧
2 (?b contains_ComplexContent_Extension ?c) ∧ (?c contains_Extension_Sequence ?d) ∧
3 (?d contains_Sequence_Element ?e) ∧ (?e ref_Element_Element ?f) ∧
4 (?f rdf:type Label−Element...) ∧ (?f value_Element_String ?range)
5 → (?domain rdf:type Variable) ∧ (?domain skos:prefLabel ?range)
```

Ontology engineers and domain experts may work together to define *SWRL rules* [154] manually only on level M1 (1) to automatically deduce OWL domain ontologies out of generated OWL ontologies on level M1 and (2) to automatically infer RDF data corresponding to domain ontologies out of RDF data conforming to generated ontologies on level M0. The *antecedents*

of SWRL rules are specified according to syntactic structures of XML documents. The *consequents* of SWRL rules, on the other side, are defined according to conceptual models of domain ontologies.

The two statements can be derived since the individual *age*, substituting the SWRL variable *?domain*, has a relationship along the object property `type_Element_Type` to an individual replacing the variable *?a*. The *?a* resource is linked to an instance *?b* via the `contains_ComplexType_ComplexContent` object property. Further, there's a navigation path from a *?b* individual to an *?f* instance through the stated object properties. As XML elements *Label*, which are instances of the EII *element* with the name 'Label' (`<xs:element name="Label"/>`), may contain text nodes such as 'Age', *?f* instances are assigned to the class *Label-Element...*, representing the EII *element* with the value 'Label' of the attribute *name*. This class assignment ensures that derived variable labels are only those strings contained in the element *Label* and not in other elements. According to the SWRL rule, *?f* resources must have at least one relationship along the data property `value_Element_String` to a *?range* literal which is substituted by the string 'Age' in the example. The concrete instances *age* and 'Age' correspond to the antecedent of the SWRL rule, i.e., there is a navigation path from the resource *age* to the string 'Age' through the stated object and data properties. Therefore, it can be inferred that the resource *age* is a variable with the variable label 'Age'.

The advantage of SWRL rules is that it works purely on the schema level and can thus be reused for any instance or document data we may encounter. By means of SWRL rules, generated ontologies' instances can be mapped to individuals of widely used and accepted ontologies like Dublin Core or SKOS. Another benefit is that all XML data conforming to XSDs can be imported automatically as domain ontologies' instances. In summary, the process of designing domain ontologies can be supplemented with all of the XSDs' information about the domains of interest, which allows us to automate part of the modeling and also to keep it closer to the original intention of the XSD.

4.5 Implementation

In this section, we accurately describe the implementation of the automatic transformations of XSDs into OWL ontologies on level M1 using XSLT [47]. The reason why we use XSLT [176] in its current version 2.0 for these transformations and therefore the contribution of this implementation approach is that we completely stay in the XML world as we only use XML technologies for (1) representing XSDs and OWL ontologies and (2) transforming XSDs into OWL by traversing XSDs' document trees. Another minor reason is that we achieve good performance when executing the XSLT stylesheet.

The second contribution is that we propose naming conventions ensuring global uniqueness of OWL equivalents of XSD constructs. The XSLT processor automatically assigns OWL class identifiers considering these naming conventions to ensure global uniqueness of IRIs. In contrast to OWL, XSD

has very few restrictions on unique naming. Hence, IRIs of OWL ontologies have to be quite long to be globally unique.

In the subsequent sub-sections, we precisely delineate the consecutive steps converting XSDs to OWL. For each step, we depict the naming conventions when translating XSD constructs into OWL constructs, describe the invoked XSLT templates and the passed parameters, and give intuitive concrete examples.

4.5.1 Definition of the RDF Document Header Information

RDF documents consist of a header and an ontology. The RDF document header information, including an XML declaration, an optional document type declaration, an RDF document header, and an OWL document header, is specified by invoking appropriate templates within the template for the document node representing the input XSD. The file extension of the output file is set to *.owl*, since OWL documents, which are RDF documents as well, are generated. The *XML declaration* includes the XML version and the character set, the encoding of the RDF document's content. The *document type declaration* contains internal entity declarations specifying abbreviations for long namespace URI strings used in the RDF document. The *RDF document header* defines the default namespace of the RDF document, the base URL, as well as the XSD, RDF, RDFS, OWL, and XSD Meta-Model Ontology's namespace prefixes. The *OWL document header* includes the statements of the ontology IRI, the ontology version IRI, and the instruction importing the XSD Meta-Model Ontology's classes, data properties, object properties, and class axioms.

The *ontology IRI* is set to `<base ontology IRI>/<local ontology IRI>.owl` and is used to identify the ontology in the context of the WWW. The ontology IRI represents the URL where the latest version of the ontology is published. The base ontology IRI is defined as *http://www.semanticweb.org/ontologies/XMLSchemaOntologies*. The local part of the ontology IRI is determined as follows: The XSD's EIIs are specified in the target namespace of the XSD. If the target namespace of the current XSD is stated, this information is used as local ontology IRI. As the target namespace is not a mandatory attribute of XSDs' root elements, a user-defined local ontology IRI can also be passed to the translation process as input parameter. Finally, if there is neither a target namespace specified nor a user-defined local ontology IRI delivered, the file name of the input XSD serves as local part of the ontology IRI.

The *ontology version IRI* corresponds to `<base ontology IRI>/<local ontology IRI>:<local ontology version IRI>.owl` and represents the URL where a given version of the ontology is published. Initially, the local part of the ontology version IRI, representing the version of the ontology the input XSD is mapped to, is set to *1.0.0*. Minor changes of the generated ontologies and the underlying XSDs will cause adjustments of the last two version levels and major ontology changes will cause modifications of the first version

stage. After the definition of the RDF document header information and the ontology, the RDF document is closed by calling an appropriate template.

4.5.2 Traversing the XML Schema's Document Tree

After the definition of the RDF document header information, the OWL ontology representing the input XSD is generated by invoking the template *ontologyDefinition*. This template serves as starting point to traverse the XSD's document tree in order to implement the transformations between the input XSD and the OWL ontology beginning with the root element of XSDs *schema*.

Each XSD EII is converted to OWL constructs by calling a template which is named according to the XSD Meta-Model Ontology's super-class representing the individual meta-EII. The template *Schema*, for instance, in called in order to create the class representing the EII *schema*. As the EII *schema* is the root element of any XSD, the XSLT processor always calls the template *Schema* first.

EIIs may contain other EIIs. Part-of-relationships are mapped to object properties `contains_<domain meta-EII>_<range meta-EII>` of the XSD Meta-Model Ontology. Thereby, containing EIIs are in the domain of these object properties and contained EIIs in their range. As, for instance, the EII *schema* may contain complex type definitions, the object property *contains_Schema_ComplexType* is defined in the XSD Meta-Model Ontology.

After calling the template *Schema*, the XSD's root element is located at the current position of the process traversing the XSD's document tree. At this time, the root element stands for the current domain EII, which may include multiple range EIIs as content. After the mapping of the current domain EII to an equivalent class of the generated ontology, the contained range EIIs have to be defined by calling appropriate templates representing the meta-EIIs they belong to. As the EII *schema* may include complex type definitions, the template *ComplexType* is invoked for each top-level complex type definition. Then, each active complex type definition stands for the current domain EII which is situated at the current position of the traversing process. When a range EII should be defined, the template named according to the appropriate meta-EII is called with the two parameters containing the name of the range meta-EII (e.g., *ComplexType*) and the name of the range EII. After the template is invoked, the range meta-EII becomes the current domain meta-EII and the range EII becomes the current domain EII, since the domain EII is now located at the current position of the XSD's document tree traversing process. If the root element of the input XSD includes the complex type definition *ComplexType1*, e.g., this complex type definition is defined by invoking the template *ComplexType* with *ComplexType* as the new domain meta-EII parameter and `ComplexType1-Type_<local ontology IRI>-Schema` as the new domain EII parameter.

The identifier of the current domain EII is built hierarchically containing all the names of the ancestor EIIs and the name of the current domain EII.

The ancestor EIIs include the current domain EII either directly or indirectly. The identifier of the XSD's root element is the rightmost part of the current domain EII's name. As a consequence, the names of the active domain EIIs are built recursively. The input XSD's identifier is set to `<local ontology IRI>-Schema` (e.g., *inputXMLSchema-Schema* for the input XSD with the target namespace *inputXMLSchema*). The containing ancestor EIIs, contributing to the overall identifier of the current domain EII, are separated by the underscore character. Each EII's associated meta-EII is also part of the entire identifier, separated from the individual EII by the minus sign. If the complex type definition called *ComplexType1*, for example, is contained in the input XSD's root element with the target namespace *inputXMLSchema*, the class with the identifier *ComplexType1-Type_inputXMLSchema-Schema* is added to the generated ontology.

4.5.3 Definition of Domain EIIs as Sub-Classes of XML Schema Meta-Model Ontology's Super-Classes

Classes, standing for domain EIIs located at the current position in the process traversing the XSD's document tree, are specified by invoking the named template *classDefinition* with the local ontology IRI and the current hierarchically built domain EII's name as formal parameters. Fully qualified identifiers of classes are determined according to the pattern `<base ontology IRI>/<local ontology IRI>.owl#<local class identifier>`. The local class identifier always refers to the name of the current domain EII. In the following, the abbreviated term *class identifier* is used, which is equivalent to the term *local class identifier*. To resolve fully qualified class identifiers, `<base ontology IRI>/<local ontology IRI>.owl#` has to be added in front of the local class identifiers.

EIIs are converted to sub-classes of the XSD Meta-Model Ontology's super-classes (`<EII>` $\sqsubseteq$ `<meta-EII>`, e.g., `Element1-Element_<local ontology IRI>-Schema` $\sqsubseteq$ `Element`) by invoking the template *superClassDefinition* with the XSD Meta-Model Ontology's super-class representing the meta-EII as parameter.

4.5.4 Definition of OWL hasValue Restrictions on XML Schema Meta-Model Ontology's Data Properties

Values of EIIs' attributes and any well-formed XML content included in the EIIs *Appinfo* and *Documentation* are transformed into XSD Meta-Model Ontology's data properties' hasValue restrictions `<domain EII>` $\sqsubseteq \exists$ `<attribute>|any_<domain meta-EII>_String.{<String>}`, as the domain EII is the sub-class of the anonymous super-class of all the individuals which have at least one relationship along the data property `<attribute>|any_<domain meta-EII>_String` to the specified individual of the primitive datatype *string*.

To define data property hasValue restrictions for the current domain EII, the named template *hasValueRestrictionOnDatatypeProperty* is invoked

with the name of the data property (e.g., *name_ComplexType_String* or *any_Documentation_String*) and the hasValue restriction, which is either the attribute value of the current domain EII (e.g., *./@name*) or the current node (.), as parameters. The presence of the optional attributes and XML contents of EIIs is checked before the data property hasValue restriction can be defined.

If hasValue restrictions on the data properties `any_<Appinfo|Documentation>_String` have to be defined, the default template for element nodes with the template mode *any* is invoked. The sequence, transmitted to this template, encompasses the current node representing the current domain EII *appinfo* or *documentation*. The element nodes `<XSD namespace prefix>:<appinfo|documentation>` may include any well-formed XML content, i.e., text nodes, element nodes, and attributes of element nodes. The XML content is added to the result tree recursively by calling the element nodes' default template with the template mode *any* for all the child and descendent element nodes of the EIIs *appinfo* and *documentation*. The element nodes *appinfo* and *documentation* themselves are not part of the output tree. In hasValue restrictions on data properties not allowed characters (<, >, ") are escaped and the attribute and text nodes for each current child or descendent element node are also appended to the output tree.

4.5.5 Definition of OWL Universal Restrictions on XML Schema Meta-Model Ontology's Object Properties

Transformation of Values of EII's Attributes Referring to other EIIs or to Type Definitions

Values of EIIs' attributes *ref*, *substitutionGroup*, and *refer*, referring to other EIIs, are converted to the XSD Meta-Model Ontology's object properties' universal restrictions `<domain EII>` $\sqsubseteq \forall$ `<ref|substitutionGroup|refer>_<domain meta-EII>_<range meta-EII>.<range EII>`. The reference to the EII *attribute* called *a1* (`<xs:attribute ref="a1"/>`), for example, is transformed into the object property universal restriction `a1-Attribute-Reference<position>_<domain EII>` $\sqsubseteq \forall$ `ref_Attribute_Attribute.a1-Attribute_<local ontology IRI>-Schema`.

Values of EIIs' attributes *type* and *base*, referring to type definitions, are transferred into XSD Meta-Model Ontology's object properties' universal restrictions `<domain EII>` $\sqsubseteq \forall$ `type|base_<domain meta-EII>_Type.<range EII>`. The value of the attribute *type* of the EII *element* named *element1* (`<xs:element name="element1" type="ComplexType1"/>`), for instance, is converted to the object property's universal restriction `element1-Element_<local ontology IRI>-Schema` $\sqsubseteq \forall$ `type_Element_Type.ComplexType1-Type_<local ontology IRI>-Schema`.

Definition of Universal Restrictions on Object Properties

Universal restrictions on these two object properties are specified by invoking the template *universalRestrictionOnObjectPropertyNotContainedRangeEle-*

ment InformationItems with the name of the object property (e.g., *ref_Attribute _Attribute* or *type_Element_Type*), the domain EII's identifier, and either the name of the range EII (e.g., *./@substitutionGroup*) or of the type definition (e.g., *./@base*), which represent the range EII part of the universal restriction, as parameters. As the attributes, including the range EIIs of the object properties' universal restrictions, are not mandatory, the object properties' universal restrictions can only be defined if the attributes are present.

In order to specify the object properties' universal restrictions, the general template *universalRestrictionOnObjectProperty* is called with the name of the object property, the range EII's ontology IRI, and the name of the range EII as parameters.

The range EIIs do not have to be defined after the specification of the object properties' universal restrictions, since classes, representing other EIIs or type definitions, are already defined or will still be defined in the process traversing the XSD's document tree.

Definition of Range EIIs' Class Identifiers

As the range EII can be defined in an external XSD, the ontology IRI of the range EII has to be determined first by invoking the named template *getNotContainedRangeElementInformationItemOntologyIRI* with the range EII as formal parameter. If the attribute includes a namespace prefix, the ontology IRI corresponds to `<base ontology IRI>/<external local ontology IRI>.owl`, since the range EII is defined in an external XSD. If the attribute does not contain a namespace prefix, the range EII is specified in the input XSD and therefore the ontology IRI is set to `<base ontology IRI>/<local ontology IRI>.owl`.

The class identifier of the range EII is determined by calling the named template *getNotContainedRangeElementInformationItemIdentifier* with the names of the range EII and the corresponding meta-EII as parameters. If an attribute includes a namespace prefix, the referenced range EII is defined in an external XSD. In this case, the range EII's identifier is set to `<range EII [without namespace prefix]>-<Range meta-EII>_<external local ontology IRI>-Schema`, since it is always referenced to top-level EIIs when they are situated in external XSDs. If, however, attributes do not contain namespace prefixes, the global range EIIs are specified in the input XSDs and therefore the identifiers of the range EIIs are set to `<range EII>-<Range meta-EII>_<local ontology IRI>-Schema`.

The EII *key*, referenced by the attribute *refer* of the EII *keyref*, is the only referenced EII which is not located at the global position of an XSD. As EIIs *key* have to be unique within an XSD, the identifiers are set as if they would be top-level EIIs: `<range EII>-Key_<local ontology IRI>-Schema`. Because of this, it can be referenced to XSD unique EIIs *key* as if they would be top-level EIIs. If referenced keys are defined in external XSDs, the target namespaces of the input XSD and of the external XSD have to be identical. Thus, the

local ontology IRI can be used to identify the external XSD in which the key is defined. As EIIs *key* are named like top-level EIIs, external XSDs do not have to be traversed to locate the EIIs containing the keys.

If type definitions, specified in external XSDs, are referenced, the type definitions' class identifiers have to be determined. The meta-EII's name is one part of the type definitions' class identifiers. If specific meta-EIIs like *SimpleType* or *ComplexType* serve as the meta-EII parts of the type definitions' class identifiers, the corresponding external XSDs, in which the type definitions are specified, have to be traversed. If, in contrast, the general meta-EII *Type* serves as the meta-EII part of the type definitions' class identifiers, the corresponding external XSDs do not have to be traversed. An obligatory traversing of external XSDs' XML document trees would be critical, since in many cases external XSDs are not physically available and namespaces can be imported using arbitrary values of *schemaLocation* attributes. Because of these reasons, the type definitions' class identifiers' meta-EII parts *AnySimpleType*, *SimpleType*, and *ComplexType* are set to *Type*. *Type* is the super-class representing the general meta-EII of the sub-classes standing for the more specific meta-EIIs of type definitions, i.e., simple ur-type, simple type, and complex type definitions.

Transformation of EIIs' Part-Of Relationships

EIIs' part-of relationships are realized by transformations into XSD Meta-Model Ontology's object properties' universal restrictions `<domain EII>` $\sqsubseteq \forall$ `contains_<domain meta-EII>_<range meta-EII>.<union of range EIIs>`. If the input XSD's root element includes only one EII *element* (e.g., `<xs:schema ...><xs:element name="element1"/></xs:schema>)`, the range of the object property can only consist of individuals of one class: `<local ontology IRI>-Schema` $\sqsubseteq \forall$ `contains_Schema_Element.element1-Element_<local ontology IRI>-Schema`. If, on the other side, EIIs like *schema* have more than one EII as content (e.g., `<xs:schema ... > <xs:element name="element1"/> <xs:element name="element2"/> </xs:schema>)`, the domain EIIs can only have relationships along the object property to individuals of the complex class consisting of the union of individuals of multiple classes representing the contained range EIIs: `<local ontology IRI>-Schema` $\sqsubseteq \forall$ `contains_Schema_Element.(element1-Element_<local ontology IRI>-Schema` $\sqcup$ `element2-Element_<local ontology IRI>-Schema)`.

For each XSD Meta-Model Ontology's object property `contains_<domain meta-EII>_<range meta-EII>` defined for the current domain EII's associated meta-EII, the template *universalRestrictionOnObjectPropertyContainedRangeElementInformationItems* is called by passing the names of the object property and of the current domain EII as parameters. As the template is called for each object property `contains_<domain meta-EII>_<range meta-EII>` of the current domain EII, it is if the checked current domain EII really includes the range EIIs of the given range meta-EII. Only in this case, the universal restriction on the object property is defined.

Definition of Range EIIs' Class Identifiers

We iterate over all range EIIs which are contained in the current domain EII in order to get all range EIIs' names. The template *getContainedRangeElementInformationItemIdentifier* is invoked for each contained range EII with the names of the range meta-EII and the current domain EII as formal parameters.

Range EIIs (1) may have an associated name, (2) may be references to top-level global EIIs, or (3) may be contained without an identifier. Range EIIs, having the attribute *name*, are named according to the pattern `<range EII>-<Range meta-EII>_<domain EII>` (`<xs:simpleType name="st3">`, e.g., is converted to `st3-Type_<domain EII>`). As EIIs *key* have to be unique within an XSD, their identifiers are determined as if they would be top-level EIIs: `<range EII>-Key_<local ontology IRI>-Schema.`

Range EIIs, which do not have an associated name, may have an attribute *ref*. If the attribute *ref* contains a namespace prefix, the referenced EII is defined in an external XSD and the input XSD's range EII's identifier is set to `<range EII [without namespace prefix]>-<Range meta-EII>-Reference <position()>-<namespace URI>_<domain EII>. <xs:attribute ref="xml:lang" />`, e.g., is transformed into `lang-Attribute-Reference<position>-http://www.w3.org/XML/1998/namespace_<domain EII>.` The namespace URI is part of the identifier, as references to top-level EIIs defined in different namespaces are possible (e.g., `<xs:attribute ref="lang"/>` and `<xs:attribute ref="xml:lang" />`). Identifiers without namespace URI statements would not be unique. Domain EIIs may include multiple range EIIs of the same meta-EII with identical *ref* attribute values (e.g., `<xs:extension...> <xs:attributeGroup ref="ag1"/> <xs:attribute Group ref="ag1"/> </xs:extension>`). To ensure the uniqueness of range EIIs' identifiers, their positions within the domain EII have to be part of their identifiers. If the attribute *ref* does not include a namespace prefix, the referenced EII is specified in the input XSD and the name of the input XSD's referencing EII is determined as `<range EII>-<Range meta-EII>-Reference<position()>_<domain EII> (<xs:attribute ref="a3"/>`, e.g., is translated into `a3-Attribute-Reference<position>_<domain EII>).`

As domain EIIs may include multiple range EIIs of the same meta-EII having no attributes *name* or *ref*, these range EIIs are identified using sequential numbers: `<Range meta-EII><position()>-<Range meta-EII>_<domain EII>` (e.g., `<xs:schema> <xs:annotation/> <xs:annotation/> </xs:schema>` is transformed into `Annotation1-Annotation_<local ontology IRI>-Schema and Annotation2-Annotation_<local ontology IRI>-Schema).` If simple or complex type definitions are included, the first range meta-EII part of the identifier is set to the more specific type definition *SimpleType* or *ComplexType* and the second range meta-EII part of the name is set to the super-class representing the more general type definition *Type* (e.g., `SimpleType1-Type_<domain EII>`) in order to distinguish simple and complex type definitions.

A variable stores the string sequence of IRIs of the ontologies in which the individual range EIIs of the given range meta-EII are defined. To realize this, the template *getContainedRangeElementInformationItemOntologyIRI* is called for each range EII. As contained range EIIs are always specified in the input XSD, the ontology IRI is determined as `<base ontology IRI>/<local ontology IRI>.owl`.

Definition of Universal Restrictions on Object Properties

To specify the universal restriction on the given object property, the XSLT processor invokes the general template *universalRestrictionOnObjectProperty* with the name of the object property and the two string sequences consisting of the range EIIs' identifiers and the associated ontology IRIs as parameters. As the active domain EII may contain one or multiple range EIIs of the same meta-EII, the class corresponding to the current domain EII can only have `contains_<domain meta-EII>_<range meta-EII>` relationships with one specific class representing a range EII or with a union of specific classes standing for the contained EIIs.

Definition of Range EIIs

As contained EIIs are referenced in definitions of universal restrictions on the object properties `contains_<domain meta-EII>_<range meta-EII>`, the range EIIs have to be defined as well by invoking the template *rangeElementInformationItemsDefinition* with the object property's name and the domain EII's identifier as parameters. For each range EII of the given range meta-EII (e.g., for each global complex type definition contained in the input XSD's root element *schema*), the range EII's identifier is determined as shown before and the template named after the range meta-EII is invoked recursively in order to define each range EII of the given meta-EII. The range meta-EII and the range EII serve as parameters. After the template is invoked, the passed range EII then becomes the current domain EII, which is now at the current position in the process traversing the XSD's document tree.

4.6 Evaluation

We provide a comprehensive evaluation[4] of our meta-model based approach designing domain ontologies based on already existing XSDs and thereby verify the subsequent hypothesis. In a technical report, we delineate the details of the evaluation [48].

Hypothesis *The effort and the time needed to deliver high quality domain ontologies from scratch by reusing information of already existing XML Schemas adequately describing particular domains is much less than creating domain ontologies completely manually and from the ground up.*

[4] Evaluation results on GitHub

4.6.1 Automatic Transformations of XML Schemas into OWL

The first step is to transform XSDs into OWL on the schema level M1 completely automatically using an XSLT stylesheet. To prove the generality of these transformations, we show that any XSD can be converted to OWL. We derived a complete set of generic test cases from the XSD meta-model in order to cover all possible and complex cases when transforming XSDs into OWL. By means of these generic test cases, we verify that any XSD can be translated into an OWL ontology and that XML documents corresponding to XSDs can be converted to an RDF representation conforming to generated ontologies.

In addition, we converted XSDs of six widely known, accepted, and applied XML standards from the academic and the industrial field: (1) *DDI-Lifecycle* [94] (see Section 3.2), (2) the *Dublin Core Metadata Element Set* [104] to offer basic cataloging information and improved document indexing for search engines (3) the *DCMI Metadata Terms* [103] to increase the specificity of metadata enabling searches to be more specific, (4) *KML* [323], an XML language focused on geographic visualization, (5) the *Atom Syndication Format (Atom)* [240], an XML-based document format that describes lists of related information known as *feeds*, and (6) the *Annotation and Image Markup Project (AIM)* [78] to add information to images in a clinical environment.

We translated 10,000 XSD constructs contained in 20 XSDs of the DDI-Lifecycle XML standard in only 30 seconds, the XSD of the Dublin Core Metadata Element Set with its 40 XSD constructs is transformed in one second, and the five XSDs of the DCMI Metadata Terms containing 250 XSD constructs in 7 seconds. All calculations can be made in under a minute. The effort in computing time is negligible in comparison with the time needed for the second step of the semi-automatic transformation process.

4.6.2 Semi-Automatic Derivation of OWL Domain Ontologies

The second step is to define SWRL rules on the schema level M1 by hand in order to derive OWL domain ontologies automatically out of generated ontologies on the schema and the instance level. We specified SWRL rules for three domain ontologies - two from the industrial and one from the academic area: (1) the Dublin Core Metadata Element Set, (2) the DCMI Metadata Terms, and (3) DDI-RDF. We defined all SWRL rules for the Dublin Core Metadata Element Set. For the DCMI Metadata Terms and DDI-RDF, we specified for each type of SWRL rules a significant amount of representative SWRL rules. For DDI-RDF, we estimated 15 person-hours to define 200 SWRL rules. As these SWRL rules are written by hand, a graphical user interface could assist users to create SWRL rules semi-automatically which would lead to time improvements.

To verify the hypothesis, we (1) determined the effort and expenses for the traditional manual approach and (2) estimated the effort and expenses for

the suggested semi-automatic approach. DDI-RDF serves as domain ontology, since we were part of the process creating it manually from scratch.

For the evaluation of the semi-automatic approach, we distinguish the time needed (1) for the formalization of the ontology and (2) to develop its conceptual model in advance. As we can reuse well-elaborated domain knowledge contained in multiple XSDs and as we can see from our experience with DDI-RDF, the effort required for the development of the conceptual model following the semi-automatic approach would be 50 percent of the working time spent for the traditional manual approach, i.e., 95 person-days or 17,750 euros[5] would have to be invested to evolve the ontology's conceptual model. For the formalization of DDI-RDF, i.e., the definition of OWL axioms and SWRL rules, we would have to invest nearly 2 person-days or 350 euros. In total, we would have to spend 97 person-days or about 18,000 euros to design the DDI-RDF domain ontology based on already available XSDs which are part of the DDI-Lifecycle XML standard. In addition, travelling, lodging, and board expenses have to be invested which allows domain experts to come together to discuss conceptual ideas. We calculate 20,000 euros for these expenses which is the half of the travelling, lodging, and board expenses spent for the traditional manual approach.

In total, 38,000 euros would be needed to design DDI-RDF using the semi-automatic approach. In contrast, the total expenses creating DDI-RDF by hand are 75,000 euros including working times as well as travelling, lodging, and board expenses. The result of the evaluation is that for the proposed semi-automatic approach we need only half of the monetary amount which therefore verifies the hypothesis.

4.7 Conclusion

To directly validate XML using common RDF validation tools against semantically rich OWL axioms when applying them in terms of constraints and extracting them from XML Schemas properly describing particular domains, we propose on formal logics and the XML Schema (XSD) meta-model based automatic transformations of arbitrary XSDs and conforming XML documents into OWL ontologies and corresponding RDF data. This does not cause any additional manual effort as these constraints have already been defined by the XML community within underlying XSDs.

OWL axioms are extracted out of XSDs either (1) explicitly according to XSD constructs determining the syntactic structure of sets of XML documents or (2) implicitly according to the implicit semantics of XSD constructs. To model explicit and implicit semantics in a semantically correct way, we formally underpin transformations themselves on semantically rich OWL axioms for which Description Logics provides the foundational basis.

As we base these transformations on the XSD meta-model, we are able to convert arbitrary complex structures of XSDs using identical transformation

[5] In the following we state rounded monetary values.

rules without any information loss. As XSDs are commonly used to specify structural relationships of objects in data-centric XML documents, we preserve all the structural information of XSD constructs.

The approach aims to speed up the task developing high quality domain ontologies from the ground up. XML Schemas serve as a basis since all the contained information about a particular domain is reused. Although RDF representations of generated ontologies can directly be validated against extracted constraints, our idea is to automatically derive more sophisticated domain ontologies and conforming RDF using manually created SWRL rules.

Generated ontologies are not directly as useful as manually created domain ontologies as they are not conform to the highest quality requirements of more demanding domain ontologies regarding the intended semantics of given domains, since XSDs only transport information about (1) the syntactic structure of sets of XML documents, (2) the terminology of individual domains, and (3) implicit semantics of XSD constructs with limited capabilities. By deriving domain ontologies, however, we (1) reduce the complexity of generated ontologies and thus underlying XSDs and (2) further supplement OWL axioms with additional domain-specific semantic information not or not satisfyingly covered by XSDs.

Acknowledgements

Chapter 4 is based on 5 publications (1 journal article, 2 articles in conference proceedings, 1 article in workshop proceedings, and 1 technical report). In [38, 46], we describe (1) how the XSD meta-model is mapped to an OWL ontology representing the XSD meta-model in OWL on the meta-model level and (2) how XSDs are mapped to OWL ontologies on the schema level in conformance with the XSD meta-model. In [47], we meticulously explain the implementation of the mapping of XSDs to OWL using XML technologies. In a journal article, we provide the formal basis of the suggested approach by means of Description Logics constructs [49], present its application in form of a complete case study, and summarize the findings of the evaluation, which we depict in detail in [48].

5

RDF Validation Requirements and Types of Constraints on RDF Data

In 2013, the W3C organized the *RDF Validation Workshop* [202], where experts from industry, government, and academia presented and discussed first case studies for constraint formulation and RDF data validation. Partly as follow-up to the workshop and partly due to further expressed requirements, two working groups on RDF validation have been established in 2014 to develop a language for expressing constraints on RDF data and defining structural constraints on RDF graphs: the *W3C RDF Data Shapes Working Group* and the *DCMI RDF Application Profiles Task Group*.

Our work is supposed to lay the ground for subsequent activities in these working groups. We propose to relate existing solutions to case studies and use cases by means of requirements, extracted from the latter and fulfilled by the former. We therefore collected the findings of the workshop and the working groups and initiated a community-driven database of requirements to formulate constraints and validate RDF data against constraints. Additionally, we added requirements from other sources, particularly in the form of constraint types that are supported by existing approaches, e.g., those expressible in OWL 2. 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.[1] The database is publicly available at http://purl.org/net/rdf-validation, continuously extended, and open for further contributions. We implemented the database using the widely adopted open source content management system Drupal.[2]

Based on our work in these working groups and jointly identified requirements, we have published by today 81 types of constraints that are required by various stakeholders for data applications, that must be expressible by constraint languages to meet all commonly approved requirements, and which

[1] At the time of the publication of this thesis, the database encompasses 17 case studies, 65 use cases, 216 requirements, and 15 solutions. The database also contains requirements not (yet) approved by the W3C and DCMI working groups.

[2] https://www.drupal.org

form the basis of this thesis. 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 (see Chapter A of the appendix [135]). In a technical report, we provide additional examples for each constraint type represented in different constraint languages [51].

We use this collection of constraint types to gain a better understanding of the expressiveness of existing solutions, i.e., to evaluate to what extent each requirement is fulfilled by the most common constraint languages (see Chapter B of the appendix [135]). We highlight strengths and weaknesses of these languages, 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 [134] and the *SPARQL Inferencing Notation (SPIN)* [185], a means to represent SPARQL in RDF, are powerful and widely used for constraint formulation and RDF data validation. With its direct support of validation via SPARQL, SPIN is very popular to check constraints [115]. The power of SPIN is shown in Table 5.1, in which we list in percent (and absolute numbers in brackets) how many constraint types each of the most common constraint languages enables to express. We consider SPIN and SPARQL as low-level implementation languages, in contrast to high-level constraint languages where specific language constructs exist to define constraints in a declarative and in comparison more intuitive and concise way – although SPARQL aficionados might object particularly to the latter point.

Table 5.1. Constraint Type Specific Expressivity of Constraint Languages

DSP	ReSh	ShEx	SHACL	OWL 2	SPIN (SPARQL)
17.3 (14)	25.9 (21)	29.6 (24)	51.9 (42)	67.9 (55)	**100.0 (81)**

We further see that OWL 2 is currently the most expressive high-level constraint language, at least according to the pure number of constraint types supported. This does not preclude that other languages are better suited for certain applications, either because they support some types that cannot be expressed by OWL 2 or because the constraint representation is more appealing to the data practitioners – producers as well as consumers who might have different needs and preferences.

It is not surprising that, at the time the thesis is published, only half of the constraint types are supported by the *Shapes Constraint Language (SHACL)* [186], a language for describing and constraining the contents of RDF graphs. The reason for this is that the development of SHACL is not yet finished. The W3C working group started developing the language at the end of 2014 and plans to publish SHACL as a W3C recommendation in June 2017.[3] We

[3] http://www.w3.org/2014/data-shapes/charter

assume and expect that SHACL will cover the majority of the constraint types at the time the language is published. In addition to the high-level vocabulary SHACL provides, so-called *native constraints* can be defined using SPARQL and similar execution languages like JavaScript. Native constraints in a language like SPARQL typically provide a lot of flexibility enabling to formulate constraints of constraint types that are only expressible by plain SPARQL. With such an extension mechanism, the combination of SHACL and SPARQL enables to express each constraint type.

In a heterogeneous environment like the Web, there is not necessarily a one-size-fits-all solution, especially as existing solutions should rather be integrated than replaced, not least to avoid long and fruitless discussions about the "best" approach. The evaluation on the constraint type specific expressivity of the most common constraint languages reveals that none of the current solutions, we consider as high-level constraint languages, satisfies all identified requirements and that rather different solutions cover distinct sets of requirements.

In Section 5.1, we depict why to take case studies and use cases as a starting point when developing standards and software in general and constraint languages in particular, why requirements form the basis for development, and why to use them to evaluate existing solutions. Subsequently, we show in form of intuitive and representative examples how constraints of different types are expressed by current constraint languages (see Section 5.2).

5.1 From Case Studies to Solutions (and Back)

In the development of standards, as in software, case studies and use cases are usually taken as a starting point. In *case studies*, the full background of a specific scenario is described, where the standard or the software is to be applied. *Use cases* are smaller units where a certain action or a typical user inquiry is described. They can be extracted from and thus linked to case studies, but often they are defined directly.

Requirements are extracted from use cases. They form the basis for development and are used to evaluate existing *solutions*. Via requirements, solutions get linked to use cases and case studies and it becomes visible which solutions can be used to meet the use cases in a given scenario and what drawbacks might be faced. Figure 5.1 visualizes the conceptual model underlying the database.

Table 5.2 shows a simplified excerpt of the database. The general structure is a polyhierarchy from case studies over use cases and requirements to solutions. All instances contain at least uplinks to the next level, i.e., solutions are linked to requirements that they fulfill and possibly requirements that they explicitly do not fulfill. Requirements are linked to use cases, which are linked to case studies.

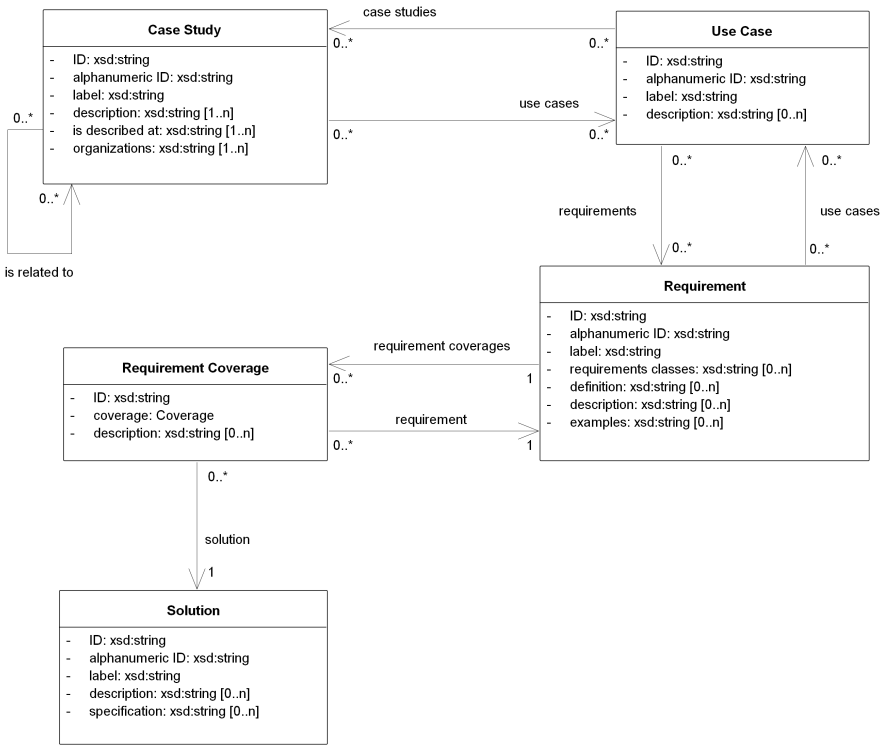

Fig. 5.1. Conceptual Model of the RDF Validation Database

Table 5.2. Example Content of the RDF Validation Database

ID	Title	Links	Description
Case Studies			
CS-1	DPLA	UC-1	The Digital Public Library of America maintains an access portal to digitized cultural heritage objects...
Use Cases			
UC-1	Recommend Property	CS-1	Some properties may not be mandatory, but may be recommended...
Requirements			
R-1	Optional Properties	UC-1	A property can be marked as optional. Valid data may contain the property.
R-2	Recommended Properties	UC-1, R-3	An optional property can be marked as recommended. A report of missing recommended properties is generated. If R-2 is fulfilled, then R-3 is fulfilled.
R-3	Classified Properties	UC-1	A custom class like “recommended” or “deprecated” can be assigned to properties and used for reporting.
Solutions			
S-1	ShEx	R-1/2/3	Fulfilled: R-1 (minimum cardinality = 0, maximum cardinality = 1). Not fulfilled: R-2, R-3.
S-2	SPIN	R-1/2/3	Fullfilled: R-1, R-2, R-3.

The polyhierarchy allows the linking of all elements to more than one parent, requirements particularly are linked to several use cases. Our goal is to maintain a set of distinct requirements. Only this way it is possible to evaluate the solutions regarding their suitability for the use cases and case studies. Use cases can be shared between case studies as well, but this is harder to maintain as use cases are less formal and often more case specific than a requirement.

Requirement *R-2* is an example, where a link between requirements is established. In this case, the link is used to point to a requirement that is broader than the more specific requirement, i.e., should requirement *R-2* be fulfilled, then requirement *R-3* is automatically fulfilled as well. In a similar way requirements can be linked to duplicates if they should occur. The goal is a relative stability regarding the requirements, which then can prove to be useful to mediate between data and solution providers.

5.2 Overview of Constraint Types

Collaboratively collected case studies and use cases led to the definition of requirements on constraint formulation and RDF data validation. Based on this comprehensive set of commonly approved requirements, we hitherto have published 81 types of constraints that are required by various stakeholders for data applications and from which concrete constraints are instantiated to be checked on the data. Each constraint type corresponds to a particular requirement in the database. In this section, we show in form of concrete, intuitive, and representative examples how constraints of different types are expressed by current constraint languages.

5.2.1 Cardinality Restrictions

Class expressions in OWL 2 can be formed by placing restrictions on the cardinality of object and data property expressions [27]. Such class expressions contain those individuals that are connected by a property expression to at least (*minimum qualified (R-75) / unqualified (R-81) cardinality restrictions*), at most (*maximum qualified (R-76) / unqualified (R-82) cardinality restrictions*), and exactly (*exact qualified (R-74) / unqualified (R-80) cardinality restrictions*) a given number of instances (of a specific class expression, in case of a qualified cardinality restriction). Qualified and unqualified cardinality restrictions can be expressed by OWL 2, ShEx, DSP, SHACL, SPIN, SPARQL, and to a limited extend ReSh.

The constraint type *minimum qualified cardinality restrictions* corresponds to the requirement *R-75*. Requirements are identified in the database by an R and a number, additionally an alphanumeric technical identifier is provided to ensure global uniqueness of requirements.[4] The *minimum qualified cardinality*

[4] In this case *R-75-MINIMUM-QUALIFIED-CARDINALITY-RESTRICTIONS*

restrictions constraint type can be instantiated to formulate the constraint that publications must have at least one author which must be a person. This constraint can be expressed as follows using different constraint languages:

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

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

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

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

SHACL: :PublicationShape
          a sh:Shape ;
          sh:scopeClass :Publication ;
          sh:property [
              sh:predicate :author ;
              sh:valueShape :PersonShape ;
              sh:minCount 1 ; ] .
          :PersonShape
              a sh:Shape ;
              sh:scopeClass :Person .

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

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

Note that the SPIN representation of the constraint is used to directly check the constraint on RDF data using a SPARQL CONSTRUCT query that creates a constraint violation triple if the constraint is violated.

The constraint type *Language tag cardinality (R-48, R-49)* is used to restrict data properties to have a minimum, maximum, or exact number of relationships to literals with selected language tags. One of the SPARQL templates provided by [192] can be used, e.g., to check that no language is used more than once per property. Thereby, *P1* is the property pointing to the literal and *V1* is the language we want to check:

```
DQTP:
SELECT DISTINCT ?s
WHERE { ?s %%P1%% ?c
    BIND ( lang(?c) AS ?l )
    FILTER (isLiteral (?c) && lang(?c) = %%V1%%)}
GROUP BY ?s HAVING COUNT (?l) > 1
```

In most library applications, *cardinality shortcuts (R-228)* tend to appear in pairs, with *optional / mandatory* establishing minimum cardinality restrictions on properties and *repeatable / non-repeatable* for maximum cardinality restrictions (see Table 5.3 for the possible pairs of cardinality shortcuts).

Table 5.3. Cardinality Shortcuts

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

5.2.2 Disjointness, Exclusivity, and Inclusivity

With the constraint type *disjoint classes (R-7)*, one can state that all of the selected classes are pairwise disjoint; that is, no individual can be at the same time an instance of a pair of these classes. Nothing can be a book and a journal article at the same time, is an example constraint of this type:

```
OWL 2 (Turtle):
:Book owl:disjointWith :Journal-Article .
```

The constraint type *disjoint properties (R-10)* enables to define that given properties are pairwise disjoint; that is, no individual x can be connected to an individual y by each pair of these properties - *author* and *title* in the subsequent example:

```
OWL 2 (Turtle):
:author owl:propertyDisjointWith :title .
```

Constraints of the type *context-specific exclusive or of property groups (R-13)* restrict individuals of given classes to have all properties of exactly one of multiple mutually exclusive property groups. Publications, e.g., are either identified by an ISBN and a title (for books) *or* by an ISSN and a title (for periodical publications), but it should not be possible to assign both identifiers to a given publication:

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

As *Moby-Dick* is a publication with an ISBN and a title without an ISSN, *Moby-Dick* is considered as a valid publication. By building rather complex disjoint unions of intersections of class expressions, we are able to construct *context-specific exclusive or of property groups* constraints in OWL 2:

```
OWL 2 (Turtle):
[   rdf:type owl:Class ;
    owl:disjointUnionOf (
        [   rdf:type owl:Class ;
            owl:intersectionOf (
                [   rdf:type owl:Restriction ;
                    owl:minQualifiedCardinality 1 ;
                    owl:onProperty :isbn ;
                    owl:onClass xsd:string ]
                [   rdf:type owl:Restriction ;
                    owl:qualifiedCardinality 1 ;
                    owl:onProperty :title ;
                    owl:onClass xsd:string ] ) ]
        [   rdf:type owl:Class ;
            owl:intersectionOf (
                [   rdf:type owl:Restriction ;
                    owl:minQualifiedCardinality 1 ;
                    owl:onProperty :issn ;
                    owl:onClass xsd:string ]
                [   rdf:type owl:Restriction ;
                    owl:qualifiedCardinality 1 ;
                    owl:onProperty :title ;
                    owl:onClass xsd:string ] ) ] ) ] .
```

An OWL 2 *disjoint union axiom* states that a class C is a disjoint union of the class expressions CE_i, $1 \leqslant i \leqslant n$, all of which are pairwise disjoint. The extensions of all CE_i exactly cover the extension of C. Thus, each instance of C is an instance of exactly one CE_i, and each instance of CE_i is an instance of C [27].

Constraints of the 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., that individuals within that context must have all property links of at least one of the inclusive property groups. The context may be a shape, an application profile, or in most cases a class. A person, e.g., must have either a name *or* at least one given name and a 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 and a family name are stated.

```
OWL 2 (Turtle):
[   rdf:type owl:Class ;
    owl:unionOf (
        [   rdf:type owl:Restriction ;
            owl:qualifiedCardinality 1  ;
            owl:onProperty foaf:name ;
            owl:onClass xsd:string ]
        [   rdf:type owl:Class ;
            owl:intersectionOf (
                [   rdf:type owl:Restriction ;
                    owl:minQualifiedCardinality 1 ;
                    owl:onProperty foaf:givenName ;
                    owl:onClass xsd:string ]
```

```
                [   rdf:type owl:Restriction ;
                    owl:qualifiedCardinality 1 ;
                    owl:onProperty foaf:familyName ;
                    owl:onClass xsd:string ] ) ] ) ] .
```

Although this kind of constraints can be realized in OWL 2, its definition is not that intuitive and declarative. Exactly the same constraint with the same meaning can be expressed more concisely with ShEx:

```
ShEx:
Person {
  ( foaf:name xsd:string
    |
    foaf:givenName xsd:string+ ,
    foaf:familyName xsd:string ) }
```

5.2.3 Constraints on RDF 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 - that is, for each individual i_1 of the class C, 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 and a clash is caused if the book *Huckleberry-Finn*, e.g., has more than one title. The data property *isbn* is functional, since books can only have one ISBN.

```
OWL 2 (Turtle):
:title a owl:FunctionalProperty .
```

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 - 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.

```
OWL 2 (Turtle):
adms:identifier a owl:InverseFunctionalProperty .
```

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, so that a system can enforce uniqueness. Books, e.g., are uniquely identified by their ISBN, i.e., the property *isbn* is inverse functional. 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 [279]. 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, Description Logics is extended with a special construct *keyfor* [211] which is implemented by the OWL 2 *hasKey* construct:

```
OWL 2 (Turtle):
:Book owl:hasKey ( :isbn ) .
```

Sub-properties (R-54/64) 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.

```
OWL 2 (Turtle):
:editor rdfs:subPropertyOf :creator .
```

Object property paths (R-55) (or *object property chains*) 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 `authorOf` $\circ$ `genre` $\sqsubseteq$ `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.

```
OWL 2 (Turtle):
authorOfGenre owl:propertyChainAxiom ( :authorOf :genre ) .
```

5.2.4 Constraints on RDF Objects

It is a common requirement to narrow down the value space of properties by an exhaustive enumeration of *allowed values* - literals (*R-37*) or objects (*R-30*). This is often rendered in drop down boxes or radio buttons in user interfaces. A constraint of this type ensures that all instances of a given class C 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 simple constraint of that type stating that books on the topic computer science can only have "Computer Science", "Informatics", and "Information Technology" as subjects:

```
SPARQL:
SELECT ?subject ?predicate ?literal
WHERE {
```

```
    ?subject rdf:type ?subC .
    ?subC rdfs:subClassOf* :Computer-Science-Book .
    ?subject ?subOP ?literal .
    FILTER ( ?predicate = ?subOP )
    ?subOP rdfs:subPropertyOf* :subject .
    FILTER ( ?literal != "Computer Science" )
    FILTER ( ?literal != "Informatics" )
    FILTER ( ?literal != "Information Technology" ) }

DSP:
[ dsp:resourceClass :Computer-Science-Book ;
  dsp:statementTemplate [
    dsp:property :subject ;
    dsp:literalConstraint [
      dsp:literal "Computer Science" , "Informatics" , "Information Technology" ] ] ] .

ReSh:
:Computer-Science-Book a oslc:ResourceShape ;
    oslc:property [ oslc:propertyDefinition :subject ;
        oslc:allowedValues [ oslc:allowedValue
            "Computer Science" , "Informatics" , "Information Technology" ] ] .

SHACL:
:Computer-Science-Book-Shape
    a sh:Shape ;
    sh:scopeClass :Computer-Science-Book ;
    sh:property [
        sh:predicate :subject ;
        sh:allowedValues
            ( "Computer Science" "Informatics" "Information Technology" ) ; ] .

ShEx:
:Computer-Science-Book {
    :subject ( "Computer Science" "Informatics" "Information Technology" ) }

OWL 2 (Turtle):
:subject rdfs:range :Computer-Science-Book-Subjects .
:Computer-Science-Book-Subjects owl:equivalentClass [ a rdfs:Datatype;
     owl:oneOf ( "Computer Science" "Informatics" "Information Technology" ) ] .
```

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. A default value is normally used when creating resources. *Default values* for objects (*R-31*) or literals (*R-38*) of given properties p within the context of given classes C are inferred automatically when these properties are not present in the data. 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 for a certain book.

```
SPIN:
owl:Thing
    spin:rule [ a sp:Construct ; sp:text """
        CONSTRUCT {
            ?subject :isPublished true . }
        WHERE {
            ?subject a :Book .
            FILTER NOT EXISTS { ?subject :isPublished ?literal } . } """ ; ] .

ReSh:
:BookResourceShape
    a oslc:ResourceShape ;
    oslc:property [
```

```
        oslc:propertyDefinition :isPublished ;
        oslc:defaultValue true ] .

SHACL:
:BookShape
    a sh:Shape ;
    sh:scopeClass :Book ;
    sh:property [
        sh:predicate :isPublished ;
        sh:defaultValue true ; ] .
```

In some cases, resources must be members of listed controlled vocabularies. Constraints of the type *membership in controlled vocabularies* (*R-32*) guarantee that individuals of a given class are assigned to the class *skos:Concept* and 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 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.

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

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 restricts that all individuals of a given class C are connected by p to i respectively l. Books on computer science, e.g., must have the topic *Computer-Science*.

5.2.5 Constraints on RDF Literals

Constraints on RDF literals are not that significant in the Linked Data community, but they are very important in communities like the library domain. Constraints of the constraint type *literal pattern matching* (*R-44*) ensure that individuals of a given class C 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 within the context of a particular class match a given regular expression which must be a valid pattern argument for the SPARQL REGEX function. The subsequent in OWL expressed literal pattern matching constraint ensures that books can only have valid *ISBN* identifiers, i.e., strings that match a given regular expression:

```
: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 a universal quantification which ensures that all literals, to which the data property *isbn* is pointing from books, must satisfy the literal pattern matching constraint.

Constraints of the constraint type *literal ranges (R-45)* make sure that for C individuals the literal values of a data property p of a certain datatype are within the literal range [V_{min},V_{max}]. The latitude of a spatial feature has to be within the literal range of [-90,90] is a typical example constraint. Constraints of the type *negative literal ranges (R-142)*, in contrast, are used to ensure that for individuals of a given class C, the literal values of a particular data property p of a given datatype are outside a specific literal range [V_{min},V_{max}]. The longitude of a spatial feature, e.g., must not be within [181,360].

Literal value comparison (R-43) constraints ensure that, depending on the datatype of data properties, two different literals of the data properties p_1 and p_2 have a specific ordering with respect to an operator like $>$, $\geqslant$, $<$, $\leqslant$, $=$, and $\neq$. 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.

For particular data properties p within the context of given classes C, values have to be stated for predefined languages. There must be an English variable name (*skos:notation*) for each *disco:Variable* is an example of a constraint of the 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:

```
SHACL and SPARQL:
:CountryShape
    a sh:Shape ;
    sh:scopeClass :Country ;
    sh:constraint [
        sh:message "Values of the data property 'germanLabel' must be
                    literals with a German language tag!" ;
        sh:sparql """
            SELECT $this ($this AS ?subject) (:germanLabel AS ?predicate)
                   (?value AS ?object)
            WHERE {
                $this :germanLabel ?value .
                FILTER (!isLiteral(?value) || !langMatches(lang(?value), "de"))
            } """ ; ] .
```

Constraints of the type *value is valid for datatype (R-223)* enable to make sure that for C individuals, each literal of a property p is valid for its datatype DT. It has to be ensured, e.g., that each literal of a certain property is actually a date value, numeric, a string, or an integer which is allowed to be negative

or not. By means of this constraint 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. It can also be enforced that stated numbers of pages of publications must be integer values which must not be negative.

5.3 Conclusion

Our work is supposed to lay the ground for subsequent activities in the working groups on RDF validation. We propose to relate existing solutions to case studies and use cases by means of requirements. We therefore collected the findings of the RDF Validation Workshop and the working groups and initiated a community-driven database of requirements to formulate constraints and validate RDF data. Additionally, we added requirements from other sources, particularly in the form of constraint types that are supported by existing approaches, e.g., those expressible in OWL 2. 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.

Laying the ground on case studies, cooperatively collected from various data practitioners, and relating solutions to case studies and use cases by means of requirements, makes sure that (1) commonly approved requirements cover real world needs of data professionals having RDF validation related problems and (2) the further development of constraint languages is based on universally accepted requirements. By linking the requirements to existing constraint languages and validation systems, we could identify strengths and weaknesses, commonalities and differences not only intellectually, but based on reliable data.

By today, we have published 81 types of constraints that are required by various 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 a specific requirement in the database. We used this set of constraint types to gain a better understanding of the expressiveness of existing and currently developed solutions. As none of the solutions, we consider being high-level constraint languages, satisfies all requirements raised by data practitioners, different languages should be used to cover different sets of requirements.

Gaps regarding requirements should be easier to close within existing solutions. This would lead to a harmonization of existing solutions regarding their expressivity and enable translations in-between or towards a general constraint language, e.g., the translation of well-readable constraints in any language to executable SPARQL queries. The latter is especially promising in view of the fact that SPARQL is able to fulfill all requirements and is

considered by many as a practical solution to validate RDF data. SPARQL and SPIN, a means to represent SPARQL in RDF, are powerful and widely used for constraint formulation and RDF data validation. We see them as low-level implementation languages, in contrast to high-level constraint languages where specific language constructs exist to define constraints in a declarative and in comparison more intuitive and concise way. We further see that OWL 2 is currently the most expressive high-level constraint language, at least according to the pure number of constraint types supported.

Acknowledgements

Two international working groups on RDF validation have been established in 2014: the *W3C RDF Data Shapes Working Group* and the *DCMI RDF Application Profiles Task Group*. As part of the editorial board of the DCMI working group and as contributors to the W3C working group, our work is supposed to lay the ground for subsequent activities in these working groups.

We propose to relate existing solutions to case studies and use cases by means of requirements extracted from the latter and fulfilled by the former. We therefore collected the findings of the *RDF Validation Workshop* [202] and the working groups and initiated a community-driven database of requirements to formulate constraints and validate RDF data. 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. In 2014, we held a tutorial on *RDF Validation in the Cultural Heritage Community*[5] and introduced how to use the database to directly contribute to the DCMI working group.

Based on our work in the DCMI and the W3C working groups 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 corresponds to a specific requirement in the database.

Chapter 5 is based on 4 publications (1 article in conference proceedings and 3 technical reports). In [43], we introduce the database, outline and classify core requirements, and give a state-of-the-art overview of already existing solutions, i.e., constraint languages. Together with other members of the DCMI working group, we have published two deliverables describing its main contributions to the RDF validation community. In [3], we give an overview of all the RDF validation related case studies and use cases collected from a variety of data practitioners, and in [4], we depict the core requirements extracted from these use cases. In a technical report, we explain each requirement/constraint type in detail, give examples for each represented in different constraint languages, and evaluate to which extend the 81 constraint types are expressible by the most common constraint languages [51].

[5] http://dcevents.dublincore.org/IntConf/index/pages/view/rdfAP-Tutorial

6

Consistent Validation across RDF-based Constraint Languages

In a heterogeneous environment like the Web, there is not necessarily a one-size-fits-all solution, especially as existing solutions should rather be integrated than replaced, not least to avoid long and fruitless discussions about the "best" approach. For constraint formulation and RDF data validation, several languages exist or are currently developed. *Shape Expressions (ShEx)* [287], *Resource Shapes (ReSh)* [274], *Description Set Profiles (DSP)* [234], the *Web Ontology Language (OWL)* [27], the *SPARQL Inferencing Notation (SPIN)* [185], and the *SPARQL Query Language for RDF* [134] are the six most promising and widely used constraint languages that are most popular among data practitioners.

With its direct support of validation via SPARQL, SPIN is very popular and certainly plays an important role for future developments in this field. Despite the fact that OWL is arguably not a constraint language, it is widely used in practice as such under the closed-world and unique name assumptions. In addition, the W3C currently develops the *Shapes Constraint Language (SHACL)* [186], an RDF vocabulary for describing RDF graph structures.

Yet, there is no clear favorite and none of the languages is able to meet all requirements raised by data practitioners, i.e., enables to express each of the 81 identified constraint types. This is the reason why further research on the development of constraint languages is needed. As constraints of different types can be expressed with different languages, we propose to base the selection of an appropriate constraint language on the requirements to be satisfied.

SPARQL is generally seen as the method of choice to validate RDF data according to certain constraints [115]. This statement is confirmed by the facts that (1) the participants of the W3C RDF Validation Workshop[1] [202] as well as the members of the DCMI and the W3C working groups agree that SPARQL should be the language to actually execute constraint validation on

[1] http://www.w3.org/2013/09/10-rdfval-minutes

RDF data and (2) constraint validation systems such as SPIN, Stardog ICV,[2] Pellet ICV,[3] and DQTP [192] check constraints with SPARQL as execution language. We claim and provide evidence from literature that constraints of each type expressed in any RDF-based constraint language can be checked by means of the low-level implementation language SPARQL.

SPARQL and SPIN, a means to represent SPARQL in RDF, are very powerful and widely used languages for constraint formulation and RDF data validation [115]. However, constraints formulated in form of SPARQL queries are not as intuitive and understandable as one wishes them to be. Consider the following example of a simple constraint of the type *allowed values (R-30/37)* stating that books on the topic computer science can only have "Computer Science", "Informatics", and "Information Technology" as subjects:

```
SPARQL:
SELECT ?subject ?predicate ?literal
WHERE {
    ?subject rdf:type ?subC .
    ?subC rdfs:subClassOf* :Computer-Science-Book .
    ?subject ?subOP ?literal .
    FILTER ( ?predicate = ?subOP )
    ?subOP rdfs:subPropertyOf* :subject .
    FILTER ( ?literal != "Computer Science" )
    FILTER ( ?literal != "Informatics" )
    FILTER ( ?literal != "Information Technology" ) }
```

This rather complex SPARQL query checks the constraint on RDF data and returns violating triples. Yet, the constraint can be formulated much shorter and in a more declarative and intuitive way using OWL 2 axioms, when treating them as constraints, in Functional-Style [27] or RDF 1.1 Turtle syntax [262]:

```
OWL 2 (Functional-Style):
DataPropertyRange( :subject :Computer-Science-Book-Subjects )
DatatypeDefinition( :Computer-Science-Book-Subjects
    DataOneOf( "Computer Science" "Informatics" "Information Technology" ) )

OWL 2 (Turtle):
:subject rdfs:range :Computer-Science-Book-Subjects .
:Computer-Science-Book-Subjects owl:equivalentClass [ a rdfs:Datatype;
     owl:oneOf ( "Computer Science" "Informatics" "Information Technology" ) ] .
```

OWL offers very powerful knowledge representation and reasoning services. The main purpose of OWL is to infer new knowledge from existing schemata and data (reasoning) rather than to check data for inconsistencies. The fact that validation is not the primary purpose of its design has led to claims that OWL cannot be used for validating RDF data.

In practice, however, many people are familiar with OWL and its concise human-understandable concrete syntax Turtle, OWL is well-spread, and RDF-S/OWL constructs are widely used to tell people and applications about how

[2] http://docs.stardog.com/#_validating_constraints
[3] http://clarkparsia.com/pellet/icv

valid instances should look like. In general, RDF documents follow the syntactic structure and the intended semantics of RDFS/OWL ontologies which could therefore not only be used for reasoning but also for validation. Consequently, OWL can be used (1) to describe RDF data, (2) to infer new knowledge out of explicitly stated knowledge, and (3) to validate RDF data against the same expressive OWL axioms when using them in terms of constraints under the closed-world and unique name assumptions (see Chapter 2.4).

With DSP, ReSh, and SHACL, exactly the same constraint of the type *allowed values* can also be expressed more intuitively and concisely than with plain SPARQL:

```
DSP:
[ dsp:resourceClass :Computer-Science-Book ;
  dsp:statementTemplate [
    dsp:property :subject ;
    dsp:literalConstraint [
      dsp:literal "Computer Science" , "Informatics" , "Information Technology" ] ] ] .

ReSh:
:Computer-Science-Book a oslc:ResourceShape ;
    oslc:property [ oslc:propertyDefinition :subject ;
        oslc:allowedValues [ oslc:allowedValue
            "Computer Science" , "Informatics" , "Information Technology" ] ] .

SHACL:
:Computer-Science-Book-Shape
    a sh:Shape ;
    sh:scopeClass :Computer-Science-Book ;
    sh:property [
        sh:predicate :subject ;
        sh:allowedValues
            ( "Computer Science" "Informatics" "Information Technology" ) ; ] .
```

And similarly, but even shorter, the identical constraint represented by ShEx:

```
ShEx:
:Computer-Science-Book {
    :subject ( "Computer Science" "Informatics" "Information Technology" ) }
```

Compared to SPARQL, high-level constraint languages like OWL, DSP, ShEx, ReSh, and SHACL are easy to understand and enable to formulate constraints more tersely, but either lack an implementation to actually validate RDF data according to constraints expressed in these languages or are based on different implementations.

This leaves the question how to execute RDF-based constraint languages on RDF data in a consistent way, i.e., how to consistently implement the validation of RDF data against constraints of any constraint type expressed in any RDF-based language. This is necessary since (1) different implementations using different underlying technologies hamper the interoperability of constraint languages and (2) full and differing implementations of several languages are hard to maintain for solution providers.

We propose to use SPARQL as low-level implementation language: constraint types are transformed into SPARQL queries executable to validate

RDF data against constraints instantiated from these constraint types. We use SPIN, a SPARQL-based way to formulate and check constraints, as basic validation framework and present a general approach how RDF-based constraint languages can be executed on RDF data in a consistent way using SPARQL as an intermediate language. The only limitations are that (1) constraints and constraint language constructs must be representable in RDF and (2) constraint languages and supported constraint types must be expressible in SPARQL.

We claim that RDF data can be validated on constraints of each type expressed in any RDF-based language using SPARQL as low-level execution language. This claim is supported by the subsequent facts: [286] showed that constraints can be translated into non-recursive Datalog programs for validation, while [5] proved that SPARQL has the same expressive power as non-recursive Datalog programs. Therefore, data validation can be reduced to SPARQL query answering and SPARQL queries can be executed to validate RDF data against constraints of any type represented in any RDF-based language which has to be expressible in SPARQL.

Datalog [77, 157] is a declarative logic programming language that is often used as a query language for deductive databases [116, 281]. Syntactically, Datalog is a subset of *Prolog*, a high-level programming language based on formal logic and devised for artificial intelligence applications [36, 98, 237].

To demonstrate the general applicability of the approach, we completely implemented the validation of RDF data against all OWL 2 and DSP language constructs by mapping each of them to SPIN. Furthermore, we provide implementations for all constraint types that are expressible in OWL 2 and DSP as well as for major constraint types representable by ReSh and ShEx.[4] In addition, we have developed a validation environment,[5] online available at http://purl.org/net/rdfval-demo, to be used to validate RDF data according to constraints of any type expressed in arbitrary RDF-based constraint languages.

In the remainder of the chapter, we first delineate the suggested general approach providing consistent implementations for any RDF-based constraint language (see Section 6.1). In Section 6.2, we introduce DSP as a language used within the DCMI community to define constraints on RDF data. Subsequently, we describe in detail how we have implemented the constraint languages DSP and OWL 2 within our SPIN validation environment (see Section 6.3).

6.1 Validation Environment

The overall idea is that we see constraint languages as domain-specific languages, hence *domain-specific constraint languages (DSCL)*, that are trans-

[4] Implementations of constraint languages on GitHub
[5] Source code on GitHub

lated into SPARQL and executed on RDF data within our validation environment we based on SPIN. Language designers are shifting attention from general purpose languages to domain-specific languages. *General-purpose languages* like Java and C++ for programming have been the primary focus of language research for a long time. The idea was to create only one language that is better suited for programming than any other language [295].

A *domain-specific language* [112] is a small, usually declarative language, tailored to a particular kind of problem [264] and offering substantial gains in expressiveness and ease of use compared with general purpose languages for the domain in question [217]. Instead of aiming to be the best for solving any kind of computing problem, domain-specific languages aim to be particularly good for solving a specific class of problems, and in doing so they are often much more accessible to the general public than traditional general-purpose languages .

The translation of a DSCL into SPARQL queries is done once, for instance, by the designer of the DSCL, and provided in form of a *SPIN mapping* plus optional *pre-processing* instructions. From a user's perspective, all that is needed is a representation of *constraints* in the DSCL and some *data* to be validated against these constraints. All these resources are purely declarative and provided in RDF or as SPARQL queries. The actual execution of the validation process is trivial using SPIN and illustrated in Figure 6.1.

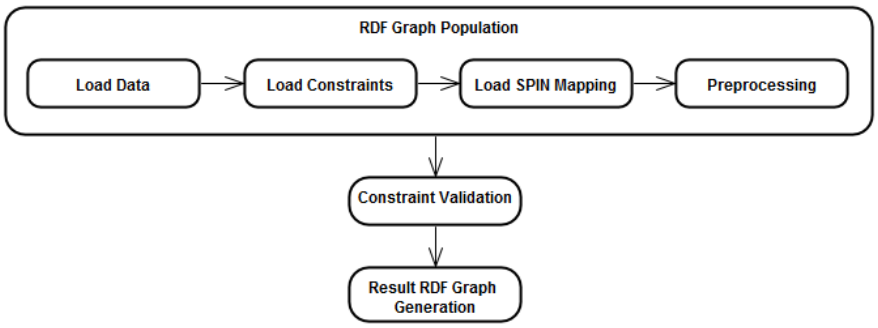

Fig. 6.1. Validation Process

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

1. The *data* is loaded that is to be validated,
2. the *constraints* in the DSCL are loaded,
3. the *SPIN mapping* is loaded that contains the SPARQL representation of the DSCL (see Section 6.3 for a detailed explanation), and
4. the *pre-processing* is performed, which can be provided in form of SPARQL CONSTRUCT queries.

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

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

6.1.1 Connect SPIN to your Data

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

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

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

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

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

6.1.2 Mapping from a DSCL to SPIN

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

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

Representing Validation Results as Constraint Violations in RDF

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

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

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

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

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

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

clause match and the SPIN construct template for checking constraints of the type *existential quantifications (R-86)* generates a violation triple.

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

Constraint violation triples may be linked to useful messages explaining how to overcome raised violations. To fix constraint violations (*spin:fix*), we may give some guidance how to become valid data either by adding, modifying, or deleting triples. To indicate how severe the violation of a constraint is, we introduce a new property to classify constraints according to different levels of severity (*severityLevel*) like *informational*, *warning*, and *error*. It is also important to find not validated triples, i.e., triples which have not been validated against any constraint, as it may be enforced that every triple of the input graph has to be validated.

Representing Constraint Checks in RDF

One of the main benefits of SPIN is that arbitrary SPARQL queries and thus constraint checks are representable as RDF triples. SPIN provides a vocabulary, the *SPIN SPARQL Syntax*, to model SPARQL queries in RDF [184]. An RDF representation of constraint checks enables that (1) they can be consistently stored together with ontologies, constraints, and the data, (2) they can easily be shared on the Web of Data, (3) they can directly be processed by a plethora of already existing RDF tools, (4) they are linkable to constraints and RDF data, and (5) the validation of RDF data can automatically be executed by SPARQL execution engines.

As for the CONSTRUCT part of the SPARQL CONSTRUCT query, SPIN also represents the WHERE clause in RDF, i.e., the actual check if constraints of a given type hold for the data. The subsequent code snippet demonstrates how SPIN represents SPARQL 1.1 NOT EXISTS [134] filter expressions in RDF (`FILTER NOT EXISTS { ?book author ?person }`) using the RDF Turtle syntax:

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

As the mapping of a DSCL to SPIN is defined for all constraint types supported by the DSCL and hence independently of concrete constraints,

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

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

6.2 DSP as Domain-Specific Constraint Language

Of course, it can be argued if DSP is the best possible way to represent constraints. DSP, however, is familiar to the DCMI community and tailored to the DCMI Abstract Model and the *Singapore Framework for Dublin Core Application Profiles* [235] comprising descriptive components that are necessary or useful for documenting application profiles. A *Dublin Core Application Profile* [86] defines metadata records which meet specific application needs. In an application profile, more than one constraint language can be used, with DSP being one of them.

A *Description Set Profile (DSP)* [234] is a DSCL to formally specify structural constraints on sets of resource descriptions within an RDF application profile. DSP restricts resources that may be described by descriptions in a description set, the properties that may be used, and the values the properties may point to.

6.2.1 Conceptual Model and RDF Mapping

The *DCMI Abstract Model* [258] with its *Description Set Model (DSM)* (see Figure 6.2) is the underlying conceptual model of Dublin Core metadata on which constraints in DSP are validated. While the DSM is highly related to RDF, it differs in some aspects. Table 6.1 shows how instances of DSM concepts are mapped to RDF. The mapping is based on *DC-RDF* [236], the recommendation how Dublin Core metadata is represented in RDF by means of the abstract syntax of the RDF model.

In the next two subsections, we delineate the concepts and their relations defined within the conceptual model of DSP and demonstrate in form of concrete, representative examples how these concepts and their interconnections are used to define constraints in DSP.

6.2.2 Constraining Resources, Properties, and Non-Literal Values

With DSP, constraints on resources can be defined within an application profile, i.e., constraints expressed in DSP enforce how valid descriptions of resources in a description set should look like. A DSP consists of a set of *dsp:DescriptionTemplate*s that put constraints on classes denoted

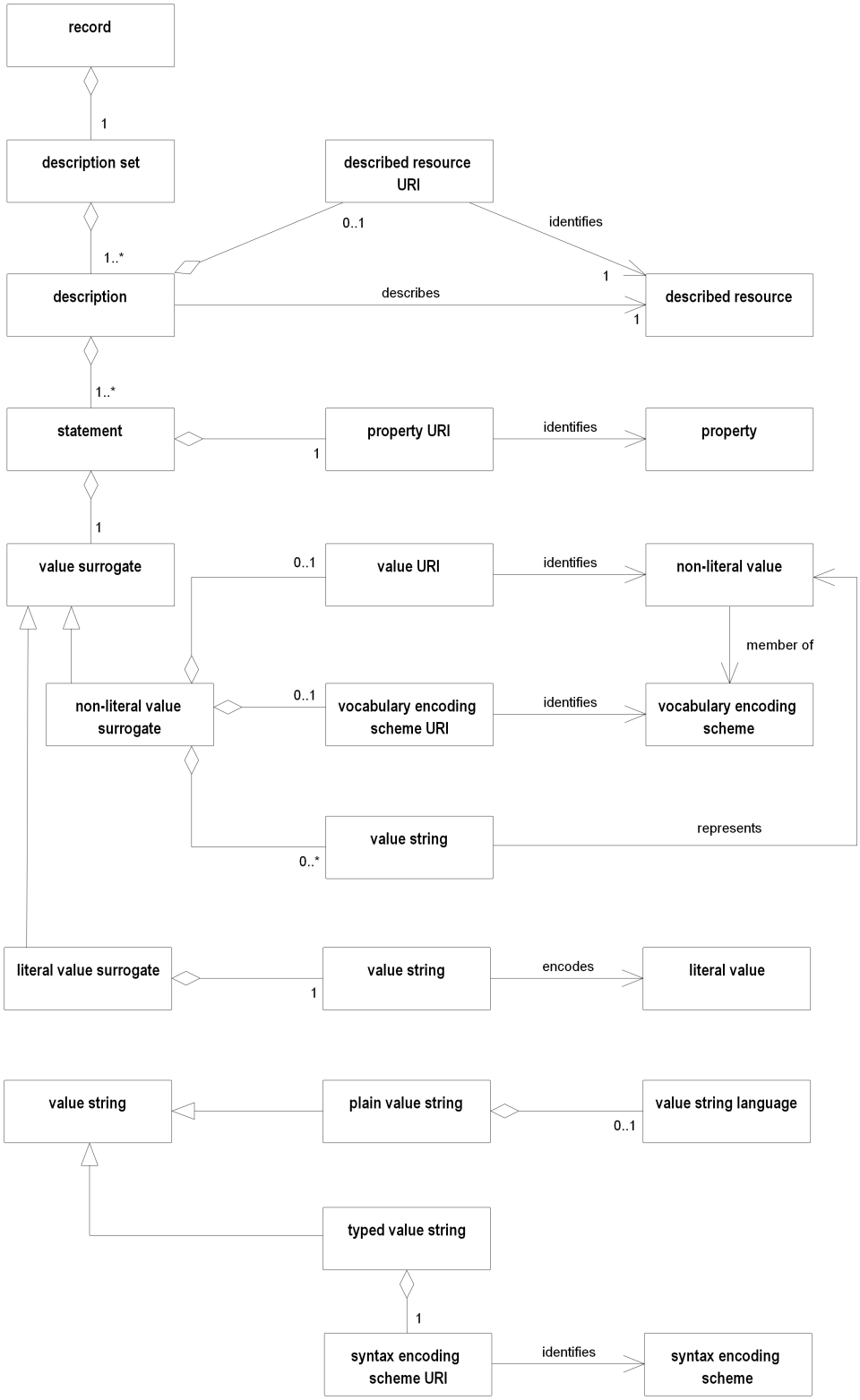

Fig. 6.2. Description Set Model (DSM)

Table 6.1. RDF Representation of Instances of DSM Concepts

DSM	RDF
record	-
description set	graph (containing description graphs)
description	graph describing a resource
resource	URI or blank node, root of description graph
statement	subject: resource predicate: property object: *value (surrogate)* (*non-literal value (surrogate)* or *literal value (surrogate))*
non-literal value (surrogate)	URI or blank node
vocabulary encoding scheme	subject: *non-literal value (surrogate)*, skos:Concept predicate: dcam:memberOf, skos.inScheme object: skos:ConceptScheme
value string	subject: *non-literal value (surrogate)* predicate: rdf:value object: literal (plain literal or typed literal)
literal value (surrogate)	literal (plain literal or typed literal)
value string language	language tag of literal
syntax encoding scheme	datatype of typed literal

by *dsp:resourceClass*. Constraints can either be associated with (1) the description of resources itself defined within *dsp:DescriptionTemplate*s, e.g., the minimum occurrence of resource class instances in the input graph, (2) single properties defined within *dsp:StatementTemplate*s, (3) non-literal values defined within *dsp:NonLiteralStatementTemplate*s, and (4) literal values defined within *dsp:LiteralStatementTemplate*s. Consider the succeeding constraints on resources, properties, and non-literal values - instantiated from the complete set of 23 DSP constraint types (see [234] for definitions, descriptions, and examples for each DSP constraint type):

```
:computer-science-book-description-template
    a dsp:DescriptionTemplate ;
    dsp:standalone true ;
    dsp:minOccur 1 ;
    dsp:maxOccur "infinity" ;
    dsp:resourceClass :Computer-Science-Book ;
    dsp:statementTemplate [
        a dsp:NonLiteralStatementTemplate ;
        dsp:minOccur 1 ;
        dsp:maxOccur 5 ;
        dsp:property :subject ;
        dsp:subPropertyOf dcterms:subject ;
        dsp:nonLiteralConstraint [
            a dsp:NonLiteralConstraint ;
```

```
            dsp:descriptionTemplate
                :computer-science-book-subjects-description-template ;
            dsp:valueClass skos:Concept ;
            dsp:valueURIOccurrence "mandatory"^^dsp:occurrence ;
            dsp:valueURI :Computer-Science, :Informatics, :Information-Technology ;
            dsp:vocabularyEncodingSchemeOccurrence "mandatory"^^dsp:occurrence ;
            dsp:vocabularyEncodingScheme :Computer-Science-Book-Subjects ;
            dsp:valueStringConstraint [
                a dsp:ValueStringConstraint ;
                dsp:minOccur 1 ;
                dsp:maxOccur 1 ;
                dsp:literal "Computer Science"@en , "Computer Science" ;
                dsp:literal "Informatics"@en , "Informatics" ;
                dsp:literal "Information Technology"en , "Information Technology" ;
                dsp:languageOccurrence "optional"^^dsp:occurrence ;
                dsp:language "en"^^xsd:language ;
                dsp:syntaxEncodingSchemeOccurrence "optional"^^dsp:occurrence ;
                dsp:syntaxEncodingScheme xsd:string ] ] ] .
```

The description template *computer-science-book-description-template* describes resources of the type *Computer-Science-Book* (*dsp:recourceClass*). *Computer-Science-Book* resources are allowed to occur standalone (*dsp:stand alone*), i.e., without being the value of a property. Books on computer science must occur at least once (*dsp:minOccur*) and may appear multiple times (*dsp:maxOccur*) in the input graph.

The DSP construct *dsp:NonLiteralStatementTemplate* is used to specify constraints on object properties with a particular resource class as domain. The *dsp:NonLiteralStatementTemplate* in the example restricts computer science books to have at least one (*dsp:minOccur*) and at most five (*dsp:maxOccur*) *subject* (*dsp:property*) relationships to non-literal values which are further restricted in the *dsp:NonLiteralConstraint*. Sub-property relationships may be enforced by *dsp:subPropertyOf*: if a computer science book is connected via *subject* to a particular topic, then this book must also be related to that topic via super-property *dcterms:subject*.

Non-literal values, to which the property *subject* is pointing, have to be of the class *skos:Concept* (*dsp:valueClass*) and are further described in a dedicated description template (referenced by *dsp:descriptionTemplate*). A URI must be given (*dsp:valueURIOccurrence* mandatory) for non-literal values, whereas allowed URIs (*dsp:valueURI*) are *Computer-Science*, *Informatics*, and *Information-Technology*. Controlled vocabularies like *Computer-Science-Book-Subjects* are represented as *skos:ConceptSchemes* in RDF and as *dsp:VocabularyEncodingSchemes* in DSM. If vocabulary encoding schemes must be stated (*dsp:vocabularyEncodingSchemeOccurrence* mandatory), they must contain the non-literal values specified within the description template. In this case, non-literal values are assigned to the class *skos:Concept* and related to the *skos:ConceptScheme Computer-Science-Book-Subjects* via the object properties *skos:inScheme* and *dcam:memberOf*. The book *Design-Patterns* satisfies all constraints defined for resources of the type *Computer-Science-Book*:

```
:Design-Patterns
    a :Computer-Science-Book ;
```

```
    :subject :Computer-Science .
:Computer-Science
    a skos:Concept ;
    dcam:memberOf :Computer-Science-Book-Subjects ;
    skos:inScheme :Computer-Science-Book-Subjects ;
    rdf:value "Computer Science" .
:Computer-Science-Book-Subjects
    a skos:ConceptScheme .
```

6.2.3 Constraining Literal Values

There are five DSP constraint types on value strings, i.e., human-readable labels of non-literal values (*rdf:value*), that can also be used for restricting literal values, as can be seen in the subsequent DSP constraints:

```
:computer-science-book-description-template
    a dsp:DescriptionTemplate ;
    dsp:standalone true ;
    dsp:minOccur 1 ;
    dsp:maxOccur "infinity" ;
    dsp:resourceClass :Computer-Science-Book ;
    dsp:statementTemplate [
        a dsp:LiteralStatementTemplate ;
        dsp:minOccur 1 ;
        dsp:maxOccur 5 ;
        dsp:property :subject ;
        dsp:literalConstraint [
            a dsp:LiteralConstraint ;
            dsp:literal "Computer Science"@en , "Computer Science" ;
            dsp:literal "Informatics"@en , "Informatics" ;
            dsp:literal "Information Technology"en , "Information Technology" ;
            dsp:languageOccurrence "optional"^^dsp:occurrence ;
            dsp:language "en"^^xsd:language ;
            dsp:syntaxEncodingSchemeOccurrence "optional"^^dsp:occurrence ;
            dsp:syntaxEncodingScheme xsd:string ] ] ] .
```

Within a *dsp:LiteralStatementTemplate*, constraints on data properties and allowed literal values for these data properties can be specified. In case the property *subject* points to literals, literal values can only be "Computer Science", "Informatics" and "Information Technology" (*dsp:literal*). For literal values of the property *subject*, the language tag *en* (*dsp:language*) and the datatype *xsd:string* (*dsp:syntaxEncodingScheme*) may be stated in the data, since *dsp:languageOccurrence* and *dsp:syntaxEncodingSchemeOccurrence* are set as optional. *Introduction-To-Algorithms* is a book on computer science fulfilling these constraints on literal values defined for resources of the type *Computer-Science-Book* and the data property *subject*:

```
:Introduction-To-Algorithms
    a :Computer-Science-Book ;
    :subject "Computer Science" .
```

6.3 Mapping of DSCLs to SPIN

After the introduction of the general approach providing consistent implementations for any RDF-based DSCL in Section 6.1, we now present concrete and intuitive examples of mappings from DSCL constructs to SPIN to offer implementations of these constructs. To demonstrate the general applicability of the approach, we created SPIN mappings for (1) all OWL 2 and DSP language constructs, (2) all of the 81 constraint types expressible by OWL 2 and DSP, and (3) major constraint types representable by ReSh and ShEx.

In the first example, we map the DSP constraint type *Statement Templates - Minimum Occurrence Constraint (6.1)* [234] to SPIN. This DSP constraint type is used to enforce the minimum number of relationships instances of given resource classes must have via given properties and corresponds to the *minimum unqualified cardinality restrictions (R-81)* constraint type. The DSP constraint type is implemented by the following SPARQL query which is then represented in SPIN-RDF and linked to the generic class *ToValidate* to ensure that all individuals of the input graph are actually validated against constraints of this type:

```
CONSTRUCT {
    _:constraintViolation
        a spin:ConstraintViolation ;
        spin:violationRoot ?subject ;
        rdfs:label ?violationMessage ;
        spin:violationSource "DSP constraint type: Minimum Occurrence Constraint" ;
        :severityLevel :error ;
        spin:violationPath ?property ;
        spin:fix ?violationFix }
WHERE {
    # detect DSP constraint type:
    ?descriptionTemplate
        dsp:resourceClass ?resourceClass .
        dsp:statementTemplate ?statementTemplate .
    ?statementTemplate
        dsp:minOccur ?minimumCardinality .
        dsp:property ?property .

    # relate DSP constraint to data:
    ?subject a ?resourceClass .

    # check constraint:
    BIND ( ( spl:objectCount ( ?subject, ?property ) ) AS ?cardinality ) .
    FILTER ( cardinality < ?minimumCardinality ) .

    # create string variables for validation results:
    BIND ( CONCAT (
        "Cardinality of '", ?property, "' ( ", ?cardinality, " ) < ",
        "mininum cardinality of '", ?property, "' ( ", ?minimumCardinality, " )" )
        AS ?violationMessage ) .
    BIND ( op:numeric-subtract( ?minimumCardinality, ?cardinality )
        AS ?cardinalityDifference ) .
    BIND ( CONCAT (
        "Add ", ?cardinalityDifference, " '", ?property,
        "' relationship(s) for '", ?subject, "'" )
        AS ?violationFix ) . }
```

It can be seen that the WHERE clause of the SPARQL CONSTRUCT query is used to detect constraint violations. First, the matching of appropriate triple patterns detects the actual DSP constraint type of a concrete constraint in DSP (lines 12-17). Second, the instance data is related to the constraint of the identified type (line 20), i.e., it is checked if the class of the currently validated RDF subject (*?subject*) of the input graph corresponds to the resource class (*?resourceClass*) as stated in the constraint. Third, it is checked if the constraint holds for the subject (lines 23-24): the SPARQL filter expression identifies only instances that violate the constraint. If the constraint does not hold, all triple patterns within the WHERE clause match, some string variables are filled as these are part of the validation results (line 27-36), and constraint violation triples are generated within the CONSTRUCT section of the SPARQL CONSTRUCT query (lines 1-9).

The example constraint creates a violation message (*?violationMessage*) that can be displayed to the user to explain the reason why the individual subject did not satisfy the constraint, together with a literal indicating the violated constraint (*spin:violationSource*) and the URIs of the subject (*spin:violationRoot*) and the properties (*spin:violationPath*) causing the violation. The property *spin:fix* is used to point to literals containing explanations of how to overcome raised violations either by adding, modifying, or deleting triples. To indicate that the language designer of DSP considers violations of constraints of that type as severe, we assign (*severityLevel*) the severity level *error* to the constraint.

In the following code snippet, we define a minimum unqualified cardinality restriction in DSP to ensure that books on computer science have at least one assigned topic (lines 2-5). As *The-C-Programming-Language* is a computer science book without any associated subject (line 8), all triple patterns within the WHERE clause match and the SPIN construct template generates an adequate constraint violation triple (lines 11-18):

```
1  DSP constraint:
2  [   dsp:resourceClass :Computer-Science-Book ;
3      dsp:statementTemplate [
4          dsp:minOccur 1 ;
5          dsp:property :subject ] ] .
6
7  Invalid RDF data:
8  :The-C-Programming-Language a :Computer-Science-Book .
9
10 Constraint violation triple:
11 [   a spin:ConstraintViolation ;
12     spin:violationRoot :The-C-Programming-Language ;
13     rdfs:label
14         "Cardinality of 'subject' ( 0 ) < minimum cardinality of 'subject' ( 1 )" ;
15     spin:violationSource "DSP constraint type: Minimum Occurrence Constraint" ;
16     :severityLevel :error ;
17     spin:violationPath :subject ;
18     spin:fix "Add 1 'subject' relationship(s) for 'The-C-Programming-Language'" ] .
```

This example demonstrates how a DSP constraint type is implemented in form of a SPARQL query within our SPIN validation framework. In the same

way, most other constraint types can be implemented as well, although the mapping often gets substantially longer and more complex. There are, however, constraint types that cannot be implemented at all, like the DSP constraint type *Literal Value Constraints - Syntax Encoding Scheme Constraint (6.5.4)* [234] which is used to determine whether syntax encoding schemes, i.e., RDF datatypes, are allowed to be stated for RDF literals.

This type of constraint cannot be implemented, since RDF literals always have associated datatype IRIs. If there is no datatype IRI and no language tag explicitly defined for an RDF literal, the datatype of the literal is by default set to *xsd:string*. If a literal has an associated language tag but no explicitly stated datatype, in contrast, the datatype is assumed to be *rdf:langString*. Fortunately, this DSP constraint type can be replaced by the equivalent DSP constraint type *Literal Value Constraints - Syntax Encoding Scheme List Constraint (6.5.5)* [234] which is used to determine the set of allowed syntax encoding schemes for literals of given data properties and which we fully implemented within the SPIN mapping for DSP.

We can specify an existential quantification in OWL 2 to make sure that any publication has at least one publisher:

```
OWL 2:
:Publication rdfs:subClassOf
  [ a owl:Restriction ;
    owl:onProperty :publisher ;
    owl:someValuesFrom :Publisher ] .
```

The next SPARQL query shows how we implemented the *existential quantifications* (*R-86*) constraint type in SPIN when expressing constraints of that type in OWL 2:

```
SPIN:
CONSTRUCT {
    _:constraintViolation
        a spin:ConstraintViolation ;
        spin:violationRoot ?subject ;
        rdfs:label ?violationMessage ;
        spin:violationSource "OWL 2 Existential Quantifications" ;
        :severityLevel :error ;
        spin:violationPath ?OPE ;
        spin:fix ?violationFix }
WHERE {
    ?subject a ?subC . ?subC rdfs:subClassOf* ?C .
    ?C a owl:Restriction ;
       owl:onProperty ?OPE ;
       owl:someValuesFrom ?CE .
    FILTER ( sp:not ( spl:hasValueOfType ( ?subject, ?OPE, ?CE ) ) ).
    BIND ( ( ... ) AS ?violationMessage ) .
    BIND ( ( ... ) AS ?violationFix ) . }
```

6.4 Conclusion

With our approach, we were able to fully implement the validation of RDF data against all OWL 2 and DSP language constructs. To demonstrate the

general applicability of the approach, we provide implementations for all constraint types which are expressible in OWL 2 and DSP as well as for major constraint types representable by ReSh and ShEx.

We use *SPIN*, a SPARQL-based way to formulate and check constraints, as a basis to create a validation environment, in which any RDF-based high-level constraint language like ShEx, ReSh, DSP, OWL, and SHACL can be implemented in a consistent way by translating them into SPARQL and executing them on RDF data.

We propose to use SPARQL as a low-level implementation language: constraint types are transformed into SPARQL queries executable to validate RDF data against constraints instantiated from these constraint types. We claim and provide evidence from literature that constraints of each type in any RDF-based language can be checked with plain SPARQL as execution language.

The translation of a constraint language into SPARQL queries is done once, for instance, by the designer of the language, and provided in form of a SPIN mapping. From a user's perspective, all that is needed is a representation of constraints in the language and some data to be validated against these constraints. All these resources are purely declarative and provided in RDF or as SPARQL queries.

The approach is particularly appealing as it has only one dependency being SPIN. The implementation of a constraint language is fully declarative, consisting of a SPIN mapping in RDF and pre-processing instructions in form of SPARQL CONSTRUCT queries which are also represented in RDF using SPIN.

We offer the developed validation environment, which is online available at http://purl.org/net/rdfval-demo, to be used to validate RDF data according to constraints of any constraint type expressed in arbitrary RDF-based languages. The SPIN engine checks for each resource if it satisfies all constraints, which are associated with the classes the resource is assigned to, and generates a result RDF graph containing information about all constraint violations.

Acknowledgements

We use SPIN, a SPARQL-based way to formulate and check constraints, as basic validation framework and propose a general approach how any RDF-based constraint language can be executed on RDF data in a consistent way using SPARQL as an intermediate language.

Chapter 6 is based on 1 article published in the proceedings of a conference. In [44], we describe the devised approach in general, give an overview of the DSP constraint language and its underlying abstract data model, and provide illustrative examples how DSP constraint types are mapped to SPIN which enables to actually validate RDF data on constraints of these constraint types expressed in DSP.

7

Validation Framework for RDF-based Constraint Languages

The evaluation on the constraint type specific expressivity of the most common constraint languages revealed that none of the current solutions, we consider as high-level constraint languages, satisfies all requirements on RDF validation, i.e., enables to express constraints of all the 81 identified constraint types (see introduction of Chapter 5)

In Chapter 6, we demonstrated that high-level RDF-based constraint languages can be implemented in a consistent way by mapping the languages to SPIN using SPARQL CONSTRUCT queries. We offer a validation environment in which own mappings from existing and newly developed constraint languages can be integrated and tested to validate RDF data according to constraints of any type expressed in these languages.

The constraint type *minimum qualified cardinality restrictions (R-75)* can be instantiated to formulate the concrete constraint that publications must have at least one author which must be a person. Equivalent constraints of that constraint type having exactly the same meaning are expressible in different languages:

```
1  OWL 2: :Publication rdfs:subClassOf
2           [ a owl:Restriction ;
3             owl:minQualifiedCardinality 1 ;
4             owl:onProperty :author ;
5             owl:onClass :Person ] .
6
7  ShEx: :Publication { :author @:Person{1, } }
8
9  ReSh: :Publication a rs:ResourceShape ; rs:property [
10          rs:propertyDefinition :author ;
11          rs:valueShape :Person ;
12          rs:occurs rs:One-or-many ; ] .
13
14 DSP: [ dsp:resourceClass :Publication ; dsp:statementTemplate [
15          dsp:minOccur 1 ;
16          dsp:property :author ;
17          dsp:nonLiteralConstraint [ dsp:valueClass :Person ] ] ] .
18
19 SHACL: :PublicationShape
20          a sh:Shape ;
21          sh:scopeClass :Publication ;
```

```
          sh:property [
              sh:predicate :author ;
              sh:valueShape :PersonShape ;
              sh:minCount 1 ; ] .
          :PersonShape
              a sh:Shape ;
              sh:scopeClass :Person .

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

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

Note that the SPIN representation of the constraint is *not* a SPIN mapping to implement its constraint type, but a direct validation of data against the constraint using a SPARQL CONSTRUCT query that creates a constraint violation in case the constraint does not hold for the data.

When we fully implemented OWL 2 and DSP and major constructs of other high-level constraint languages using SPARQL as intermediate language and mappings to SPIN, we found that many mappings actually resemble each other; particularly the mappings of the same constraint type in different languages, but also the mappings of different constraint types, though the latter only on a very superficial, structural level. In the example, it can be seen that the expressions of the high-level constraint languages are comparatively similar - there seems to be a pattern, a common way to express this type of constraints.

Creating mappings of constraint languages to SPIN to actually implement their validation is in many cases not straight-forward and requires profound knowledge of SPARQL, as the SPIN mappings for OWL 2 and DSP demonstrate. In the next example, we show how the validation of the *minimum qualified cardinality restrictions* constraint type is implemented for DSP:

```
CONSTRUCT {
    _:constraintViolation
        a spin:ConstraintViolation ;
        spin:violationRoot ?subject ;
        rdfs:label ?violationMessage ;
        spin:violationSource ?violationSource ;
        :severityLevel :error ;
        spin:violationPath ?predicate ;
        spin:fix ?violationFix }
WHERE {
    ?subject a ?resourceClass .
    ?descriptionTemplate
        dsp:resourceClass ?resourceClass ;
        dsp:statementTemplate ?statementTemplate .
    ?statementTemplate
```

```
16          dsp:minOccur ?minimumCardinality ;
17          dsp:property ?predicate ;
18          dsp:nonLiteralConstraint ?nonLiteralConstraint .
19      ?nonLiteralConstraint dsp:valueClass ?valueClass .
20      BIND ( qualifiedCardinality( ?subject, ?predicate, ?valueClass ) AS ?cardinality ) .
21      FILTER ( ?cardinality < ?minimumCardinality ) .
22      BIND ( ( ... ) AS ?violationMessage ) .
23      BIND ( ( ... ) AS ?violationSource ) .
24      BIND ( ( ... ) AS ?violationFix ) . }
```

We build on the experience gained from mapping several constraint languages to SPIN and from the analysis of the identified constraint types to create an intermediate layer, a framework that is able to describe the mechanics of all constraint types in a way that mappings from high-level constraint languages to this intermediate generic representation can be created straight-forwardly. The basic idea of our framework is very simple: (1) we aim at reducing the representation of constraints to the absolute minimum that has to be provided in a mapping to SPIN and (2) we want to be able to express constraints of any constraint type in order to meet all requirements raised by data practitioners.

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

1. *How to ensure for any constraint type that RDF data is consistently validated against semantically equivalent constraints of the same constraint type across RDF-based constraint languages?*
2. *How to ensure for any constraint type that semantically equivalent constraints of the same constraint type can be transformed from one RDF-based constraint language to another?*

1. Consistent Validation regarding Validation Results for each Constraint Type across RDF-based Constraint Languages

Even though SPIN provides a convenient way to represent constraint violations and to validate RDF data, the implementation of a high-level constraint language still requires a tedious mapping to SPIN with a certain degree of freedom how the violation of constraints of a certain type is represented and how constraints of a particular type are checked.

Checks of semantically equivalent constraints of the same constraint type should detect the same set of violations regardless of the language used to express them. This means that whenever semantically equivalent constraints in different languages are checked on RDF data they should point out the same violations.

Our framework therefore provides a common ground that is solely based on the abstract definitions of constraint types. By providing just one SPIN mapping for each constraint type,[1] we ensure that the details of the SPIN implementation are consistent for a specific constraint type irrespective of the language and that the validation of semantically equivalent constraints of the same constraint type in different languages leads always to exactly the same results.

2. Transforming Semantically Equivalent Constraints of any Constraint Type across RDF-based Constraint Languages

As there is no standard way to express constraints, semantically equivalent constraints of the same type are expressible by a variety of constraint languages - each of them different in syntax and semantics. Transformations of semantically equivalent constraints of the same type from one language to another are important (1) to enhance the interoperability of constraint languages, (2) to resolve misunderstandings and ambiguities in the interpretation of constraints, and (3) to avoid the necessity to understand several constraint languages.

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

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

Mappings from constraint languages to the abstraction layer and back enable transformations of semantically equivalent constraints of the same type from one constraint language to another and therefore increase the interoperability of constraint languages.

We do not need to define mappings for each constraint type and each possible combination of n constraint languages. Assuming that we are able to express constraints of a single constraint type like *minimum qualified cardinality restrictions* in 10 languages; $n \cdot (n-1) = 90$ mappings would be needed, as mappings generally are not invertible. With an intermediate generic representation of constraints, on the other side, we only need to define $2n = 20$

[1] Generic SPIN mappings for constraint types on GitHub

mappings for each constraint type – where 10 mappings should already exist if we have an implementation for that particular constraint type in our framework.

Furthermore, an additional abstraction layer simplifies the implementation of existing and newly developed constraint languages: all that is needed to reuse consistent implementations of constraint types, which are based on their abstract definitions, is to define bidirectional mappings between constraints specifically expressed in a particular language and generically formulated constraints against which the validation of RDF data is actually implemented.

To summarize, if language developers are willing to provide two mappings – forward (m) and backward (m') – for each supported constraint type using our framework, we would not only get the consistent implementation of all languages, it would also be possible to transform semantically equivalent constraints between all constraint languages.

The remainder of this chapter is organized in four parts. In Section 7.1, we introduce the general validation framework, its additional abstraction layer on top of SPARQL, and its core building blocks.

In Section 7.2, we delineate in form of an intuitive and representative example how a constraint of a given constraint type in a specific language is translated into its generic intermediate representation to be checked on RDF data and how to provide a consistent implementation for that particular constraint type across constraint languages.

We continue this example and depict how the generic representation of the constraint is mapped back to its specific representation in a certain language which forms the basis to transform the constraint into any other semantically equivalent constraint expressed in any other RDF-based constraint language (see Section 7.3).

In Section 7.4, we sketch how to combine the framework with SHACL, the constraint language the W3C working group currently develops, in order to (1) generate SHACL extensions for constraint types not supported by SHACL, (2) enhance the interoperability between SHACL and other constraint languages, and (3) maintain the consistency of the implementations of constraint types among SHACL and other languages.

7.1 Validation Framework

When we fully implemented OWL 2 and DSP and to some extent other constraint languages using SPARQL as intermediate language, we found that many mappings to SPIN actually resemble each other; particularly the mappings of the same constraint type in different languages, but also the mappings of different constraint types, though the latter only on a very superficial, structural level. The basic idea of our framework is very simple: (1) we aim at reducing the representation of constraints to the absolute minimum that has to be provided in a mapping to SPIN to implement the validation for

constraint types and (2) we want to be able to express constraints of any constraint type in order to meet all requirements raised by data practitioners.

Consider again our example for the SPIN representation of a concrete constraint of the type *minimum qualified cardinality restrictions (R-75)* which is checked to ensure that publications must have at least one author which must be a person:

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

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

However complex this SPIN code looks like, all we have to provide to make it work is the desired minimum cardinality (?minimumCardinality), the property to be constrained (?predicate), the class whose individuals must hold for the constraint (?C1), and the class for which the property should be restricted (?C2). All other variables are bound internally. So we could reduce the effort of the mapping by simply providing these four values, which are readily available in all representations of the constraint in different high-level constraint languages:

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

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

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

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

SHACL: :PublicationShape
          a sh:Shape ;
          sh:scopeClass :Publication ;
          sh:property [
              sh:predicate :author ;
              sh:valueShape :PersonShape ;
              sh:minCount 1 ; ] .
       :PersonShape
          a sh:Shape ;
          sh:scopeClass :Person .
```

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

7.1.1 Building Blocks

At the core, we use a very simple conceptual model for constraints (see Figure 7.1), using a small lightweight vocabulary called *RDF Constraints Vocabulary (RDF-CV)*.[2]

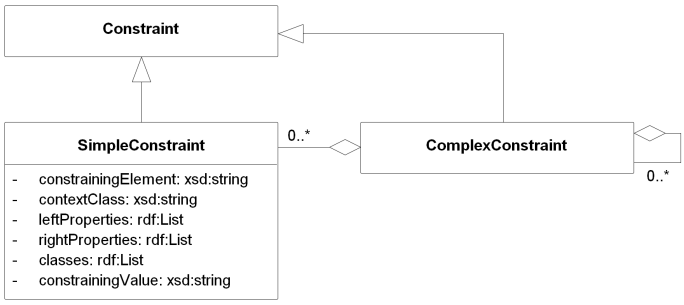

Fig. 7.1. RDF Constraints Vocabulary (RDF-CV) Conceptual Model

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 – we will come to the notion of a constraining element in a second. In contrast, there are *complex constraints*, i.e., the set of constraints which are created out of simple and/or other complex constraints. 60% of the overall 81 identified constraint types are used to instantiate simple constraints and 26% to instantiate complex constraints. Constraints of 14% of the constraint types are complex constraints which can be simplified and therefore formulated as simple constraints if additional constraining elements are introduced to cover them (see Table 7.1 and Chapter C of the appendix [135] for the individual assignments of constraints of each constraint type to constraint sets to be distinguished according to the RDF-CV).

Table 7.1. Constraint Sets

Constraint Sets	**% (#)**
Simple Constraints	60.5 (49)
Simple Constraints (Syntactic Sugar)	13.6 (11)
Complex Constraints	25.9 (21)
DL Expressible	64.2 (52)
Not DL Expressible	35.8 (29)
Total	100 (81)

[2] Formal specification, HTML documentation, and UML class diagram on GitHub

The properties describing a simple constraint are very structural, i.e., the properties describe the structure of constraints. The central property is the *constraining element* which refers to one of 103 constraining elements - described and assigned to constraint types in Chapter E of the appendix [135]. Constraining elements are, e.g., taken from Description Logics. Another example of a constraining element is the SPARQL function REGEX where a regular expression is checked against some property value. In most cases, a constraining element directly corresponds to a single constraint type, sometimes (as for REGEX) 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. Complex constraints again need several constraining elements to be formulated.

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

This simple structure plus the constraining elements form the building blocks of our proposed framework. In Chapter E of the appendix [135], 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.

Formal Approach and Semantics

A cornerstone of the framework is the generic representation of constraints, which can often be done using Description Logics, as for the *minimum qualified cardinality restriction* `Publication` $\sqsubseteq \geqslant 1$ `author.Person`. This way, the knowledge representation formalism *Description Logics (DL)* [12, 13, 196] with its well-studied theoretical properties provides the foundational basis for the framework. It turned out that constraints of 64% of the 81 constraint types are actually expressible in DL. Only for the remaining 36%, other means, i.e., other constraining elements, had to be identified in order to be able to formulate constraints of these types. This is not surprising if we consider that OWL is based on DL. In Chapter E of the appendix [135], we list for each in DL

expressible constraint type the DL constructs needed to formulate constraints of that particular type.

When we talk about using DL to represent constraints, we have to establish once more that the semantics of OWL and DL differ from the semantics of constraint languages regarding the open world assumption (OWA) and the non-unique name assumption (nUNA). Both are usually assumed when dealing with OWL or DL, whereas validation usually assumes a closed world assumption (CWA) and unique naming assumption (UNA), i.e., if a desired property is missing, this leads to a violation and if two resources are named differently, they are assumed to be different resources.

The applied semantics has to be defined if validation is performed, as the results differ under different semantics. Precisely, we found that for 56.8% of the constraint types validation results differ if the CWA or the OWA is assumed and for 66.6% of the constraint types validation results are different in case the UNA or the nUNA is assumed (see Chapter D of the appendix [135]).

For the purpose of consistent implementations and transformations of semantically equivalent constraints, constraints are considered to be *semantically equivalent* if the same set of violations are detected regardless of the language used to express them, which means whenever the constraints are expressed in different languages and checked on RDF data they point out the same violations.

7.1.2 Simple Constraints

In this and the following section, we provide examples for the representation of constraints within the framework. The *minimum qualified cardinality restriction (R-75)* `Publication` $\sqsubseteq \geqslant 1$ `author.Person`, which restricts publications to have at least one author which must be a person, is an example of a simple constraint on the property *author* which holds for all individuals of the class *Publication*. Table 7.2 displays how the simple constraint is generically represented using the RDF-CV:

Table 7.2. Minimum Qualified Cardinality Restriction as Simple Constraint

context class	left properties	right p.	classes	constraining element	c. value
Publication	author	-	Person	$\geqslant$	1

The *constraining element* is an intuitive term which indicates the actual type of constraint. For the majority of the constraint types, there is exactly one constraining element. For the constraint type *property domains (R-25)*, e.g., there is only one constraining element with the same identifier *property domain* (in singular form as the constraining element indicates one concrete constraint of this type). Some constraint types, however, need several constraining elements to be expressed; *language tag cardinality (R-48, R-49)*, e.g., is used to restrict data properties to have a minimum, maximum, or

exact number of relationships to literals with selected language tags. Thus, three constraining elements are needed to express each possible constraint of that constraint type. This example also illustrates that the granularity of the constraint types varies and certainly often is debatable. Keep in mind that they correspond to requirements as identified by various data practitioners. The constraining elements, as in this example, are closer to atomic elements of constraints.

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 such as *data property facets (R-46)* or *literal pattern matching (R-44)*, we reuse widely known terms from SPARQL (e.g., REGEX) or XML Schema constraining facets (e.g., *xsd:minInclusive*) as constraining elements. We provide a complete list of all 103 constraining elements to be used to express constraints of any constraint type (see Chapter E of the appendix [135]).

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

The assignment of properties to left or right property lists depends on the constraining element. *Object property paths* (*R-55*) ensure that if an individual x is connected by a sequence of object properties with an individual y, then x is also related to y by a particular object property. As *Stephen-Hawking* is the author of the book *A-Brief-History-Of-Time* whose genre is *Popular-Science*, the object property path `authorOf` $\circ$ `genre` $\sqsubseteq$ `authorOfGenre` infers that *Stephen-Hawking* is an author of the genre *Popular-Science*. Thus, when representing the constraint using the RDF-CV (see Table 7.3), the properties *authorOf* and *genre* are placed on the left side of the constraining element *property path* and the property *authorOfGenre* on its right side.

The *context class* limits the constraint to individuals of a specific class. A context class may be an rdfs:Class, an owl:Class (as sub-class of rdfs:Class), or an rdfs:Datatype which is both an instance of and a sub-class of rdfs:Class. As the *object property paths* constraint must hold for all individuals within the data, the context class is set to the *DL top concept* $\top$ which stands for the super-class of all possible classes.

Table 7.3. Object Property Path as Simple Constraint

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

Constraints of 36% of the constraint types are not expressible in DL, but can still be described using the RDF-CV, such as constraints of the type *literal pattern matching* (*R-44*) which restrict literals to match given patterns. The *universal quantification (R-91)* `Book` $\sqsubseteq \forall$ `identifier.ISBN` encapsulates a literal pattern matching constraint to ensure that books can only have valid *ISBN* identifiers, i.e., strings that match a given regular expression. Even though constraints of the type *literal pattern matching* cannot be expressed in DL, OWL 2 can be used to formulate 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 datatype restriction on *xsd:string*. The datatype *ISBN* can be used just like any other datatype like in the universal quantification above.

Table 7.4 presents how (1) the not in DL expressible literal pattern matching constraint and (2) the in DL expressible universal quantification are both generically represented as simple constraints using the RDF-CV. Thereby, the context class *ISBN*, whose instances must satisfy the literal pattern matching constraint, is reused within the list of classes the universal quantification refers to. The literal pattern matching constraint type introduces the constraining element *REGEX* whose validation has to be implemented once like for any other constraining element.

Table 7.4. In DL and NOT in DL Expressible Constraints as Simple Constraints

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

7.1.3 Complex Constraints

Constraints of the type *context-specific exclusive or of property groups (R-13)* restrict individuals of given classes to have property relationships of all properties of exactly one of multiple mutually exclusive property groups. Publications, e.g., are either identified by an ISBN and a title (for books) or by an ISSN and a title (for periodical publications), but it should not be possible to assign both identifiers to a given publication - in ShEx:

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

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

$$\mathbf{Publication} \sqsubseteq (\neg \mathbf{E} \sqcap \mathbf{F}) \sqcup (\mathbf{E} \sqcap \neg \mathbf{F})$$
$$E \equiv A \sqcap B \text{ , } F \equiv C \sqcap D$$
$$A \sqsubseteq \geqslant 1 \text{ isbn.string} \sqcap \leqslant 1 \text{ isbn.string}$$
$$B \sqsubseteq \geqslant 1 \text{ title.string} \sqcap \leqslant 1 \text{ title.string}$$
$$C \sqsubseteq \geqslant 1 \text{ issn.string} \sqcap \leqslant 1 \text{ issn.string}$$
$$D \sqsubseteq \geqslant 1 \text{ title.string} \sqcap \leqslant 1 \text{ title.string}$$

The 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)*).

Constraints of almost 14% of the constraint types are complex constraints which can be simplified and therefore formulated as simple constraints when using them in terms of syntactic sugar. As *exact (un)qualified cardinality restrictions (R-74/80) (=n)* and *context-specific exclusive or of property groups (R-13)* are constraint types of frequently used complex constraints, we propose to simplify them in form of simple constraints. As a consequence, the *context-specific exclusive or of property groups* complex constraint is represented more intuitively and concisely as a generic constraint by means of the RDF-CV (see Table 7.5).

Table 7.5. Simplified Complex Constraint

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

The *primary key properties (R-226)* constraint type is often useful to declare a given (datatype) property as the primary key of a class, so that a system can enforce uniqueness. Books, e.g., are uniquely identified by their ISBN, i.e., the property *isbn* is inverse functional (`funct` $isbn^-$) which can be represented using the RDF-CV in form of a complex constraint consisting of two simple constraints (see Table 7.6). The meaning of these simple constraints is that ISBN identifiers can only have $isbn^-$ relations to at most one distinct book.

Table 7.6. Primary Key Property as Complex Constraint

context class	left p.	right p.	classes	c. element	c. value
$\top$	$isbn^-$	isbn	-	inverse property	-
Book	$isbn^-$	-	-	$\leqslant$	1

Keys, however, are even more general, i.e., a generalization of inverse functional properties [279]. A key can be a datatype, an object property, or a chain of properties. For these generalization purposes, as there are different sorts of keys, and as keys can lead to undecidability, DL is extended with a special construct *keyfor* [211]. When using keyfor (`isbn keyfor Book`), the complex

constraint can be simplified and thus formulated as a simple constraint using the RDF-CV which looks like the following in concrete RDF turtle syntax:

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

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

Constraints of the *irreflexive object properties (R-60)* constraint type ensure that no individual is connected by a given object property to itself [196]. With the irreflexive object property constraint $\top \sqsubseteq \neg\exists authorOf.Self$, e.g., one can state that individuals cannot be authors of themselves. When represented using the RDF-CV, the complex constraint aggregates three simple constraints (see Table 7.7):

Table 7.7. Irreflexive Object Property as Complex Constraint

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

When using the *irreflexive object property* constraint in terms of syntactic sugar, the complex constraint can be expressed more concisely in form of a simple constraint with exactly the same semantics (see Table 7.8):

Table 7.8. Irreflexive Object Property as Simple Constraint

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

7.2 Consistent Validation regarding Validation Results across RDF-based Constraint Languages

Independently of the language used to express constraints, checks of semantically equivalent constraints should detect the same set of violations, which means whenever semantically equivalent constraints in different languages are checked on RDF data they should point out the same violations. Our framework therefore provides a common ground that is solely based on the abstract definitions of constraint types. By providing a SPIN mapping for each constraint type, it is ensured that the details of the SPIN implementation are consistent irrespective of the constraint language and that the validation of

semantically equivalent constraints of the same constraint type in different languages leads always to exactly the same results independently of the language used to express the constraints.

Using the framework for consistent implementations of constraint languages is straight-forward. For each language construct, the corresponding constraint type and constraining element have to be identified. Again we use the constraint `Publication` $\sqsubseteq \geqslant 1$ `author.Person` of the type *minimum qualified cardinality restrictions (R-75)* which is supported by OWL 2:

```
:Publication rdfs:subClassOf
    [ a owl:Restriction ;
      owl:minQualifiedCardinality 1 ;
      owl:onProperty :author ;
      owl:onClass :Person ] .
```

From Table 7.2, we know the representation of the constraint in our framework, which corresponds to the following generic intermediate representation in RDF using the RDF-CV:

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

For the constraint type *minimum qualified cardinality restrictions* and the respective constraining element, a mapping simply constructs this generic representation out of the specific OWL 2 representation of the constraint using a SPARQL CONSTRUCT query:

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

The SPIN engine executes such mappings to convert constraints in high-level constraint languages to their intermediate generic representation. The property *spin:rule* links an rdfs:Class to SPARQL CONSTRUCT queries.

Each query defines an inference rule that is applied to all instances of the associated class and its sub-classes. The inference rule defines how additional triples are inferred from what is stated in the WHERE clause. For each binding of the triple patterns within the WHERE clause of the rule, the triples in the CONSTRUCT clause are generated and added to the underlying model as inferred triples. At query execution time, the SPARQL variable *?subject* is bound to the currently validated instance of the class. As each resource per default is assigned to the class *owl:Thing*, this inference rule is evaluated for each subject of the input RDF graph.

Each constraint type and constraining element is implemented in exactly the same way by providing a SPIN mapping which encompasses SPARQL queries that check generically represented constraints on RDF data and produce constraint violation triples if constraints are violated, as described in the previous Chapter 6. The next code snippet shows how we implemented the validation of RDF data against constraints of the type *minimum qualified cardinality restrictions* in case they are generically expressed using the RDF-CV:

```
# construct constraint violation triples:
CONSTRUCT {
    _:constraintViolation
        a spin:ConstraintViolation ;
        spin:violationRoot ?subject ;
        rdfs:label ?violationMessage ;
        spin:violationSource ?violationSource ;
        :severityLevel :error ;
        spin:violationPath ?lp1 ;
        spin:fix ?violationFix }
WHERE {
    # detect constraint type and constraining element:
    [   a rdfcv:SimpleConstraint ;
        rdfcv:contextClass ?cc ;
        rdfcv:leftProperties ( ?lp1 ) ;
        rdfcv:classes ( ?c1 ) ;
        rdfcv:constrainingElement "minimum qualified cardinality restriction" ;
        rdfcv:constrainingValue ?cv ] .

    # relate constraint to data:
    ?subject a ?cc .

    # check constraint:
    BIND ( rdfcv2spin:qualifiedCardinality( ?subject, ?lp1, ?c1 ) AS ?c ) .
    BIND( STRDT ( STR ( ?cv ), xsd:nonNegativeInteger ) AS ?minimumCardinality )
    BIND( STRDT ( STR ( ?c ), xsd:nonNegativeInteger ) AS ?cardinality )
    FILTER ( ?cardinality < ?minimumCardinality ) .

    # create string variables for validation results:
    BIND ( ( ... ) AS ?violationMessage ) .
    BIND ( ( ... ) AS ?violationSource ) .
    BIND ( ( ... ) AS ?violationFix ) . }
```

7.3 Transforming Constraints across RDF-based Constraint Languages

As stated in the introduction of this chapter, we see a huge potential in the possibility to transform semantically equivalent constraints from one high-level constraint language to another via the RDF-CV representation, to avoid that for each constraint type every possible combination of constraint languages has to be mapped separately. The following SPIN inference rule exemplifies this approach and provides a mapping from the RDF-CV back to the OWL 2 constraint of the type *minimum qualified cardinality restrictions*:

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

It can be seen that the mapping is quite similar to the first mapping from OWL 2 constraints to RDF-CV constraints and basically simply switches the CONSTRUCT and WHERE part of the query, with slight adjustment in the structure of the variables. Potentially an even simpler representation for the mapping could be found that would enable the creation of forward and backward mappings out of it. We didn't investigate this further, though, and it is not yet clear if there can be cases where the backward mapping is more different.

7.4 Combining the Framework with SHACL

In this section, we sketch how to combine the framework with SHACL, the constraint language the W3C working group currently develops, in order to (1) generate SHACL extensions for constraint types not supported by SHACL, (2) enhance the interoperability between SHACL and other constraint languages, and (3) maintain the consistency of the implementations of constraint types among SHACL and other languages.

7.4.1 Generating SHACL Extensions for Constraint Types

As the Shapes Constraint Language (SHACL) [186], a language for describing and constraining the contents of RDF graphs, is still under constant development, it is not surprising that, at the time the thesis is published, SHACL supports only half of the constraint types. We assume and expect that SHACL will cover the majority of the constraint types when the language is published as W3C recommendation.

In addition to the high-level vocabulary SHACL provides, so-called *native constraints* can be defined using SPARQL and similar execution languages like JavaScript. Native constraints in a language like SPARQL typically provide a lot of flexibility which enables to formulate constraints of constraint types which are only expressible by plain SPARQL. With such an extension mechanism, the combination of SHACL and SPARQL enables to express all constraint types.

Within the vocabulary of SHACL, there is no term which allows to specify constraints of the type *language tag matching (R-47)*. Though, a native SPARQL constraint of that type may be associated with a shape to restrict that values of the data property *germanLabel* must be literals with a German language tag:

```
SHACL Shapes Graph:
:CountryShape
    a sh:Shape ;
    sh:scopeClass :Country ;
    sh:constraint [
        sh:message "Values of the data property 'germanLabel' must be
                    literals with a German language tag!" ;
        sh:sparql """
            SELECT $this ($this AS ?subject) (:germanLabel AS ?predicate)
                   (?value AS ?object)
            WHERE {
                $this :germanLabel ?value .
                FILTER (!isLiteral(?value) || !langMatches(lang(?value), "de"))
            } """ ; ] .

Data Graph:
:ValidCountry
    a :Country ;
    :germanLabel "Die vereinigten Staaten von Amerika"@de .

:InvalidCountry
    a :Country ;
    :germanLabel "The United States of America"@en .
```

The scope of the shape includes all instances of *Country*. For those instances (represented by the variable $this), the SPARQL query walks through the values of *germanLabel* and verifies that they are literals with a German language code. Exactly the same semantically equivalent constraint can be generically and concisely formulated using the RDF-CV:

```
RDF-CV:
[   a rdfcv:SimpleConstraint ;
    rdfcv:contextClass :Country ;
```

```
    rdfcv:leftProperties ( :germanLabel ) ;
    rdfcv:constrainingElement "language tag matching" ;
    rdfcv:constrainingValue "de" ] .
```

In the same way, each constraint type without a corresponding term in the SHACL vocabulary is generically expressible by means of the RDF-CV. It is conceivable to derive SHACL extensions with a SPARQL body (like the one above) for each constraint type not supported by SHACL out of the generic intermediate representation of constraint types and their SPIN mappings including the SPARQL queries to actually check instantiated constraints.

7.4.2 Enhancing the Interoperability between SHACL and other Constraint Languages

Mappings from SHACL and other high-level constraint languages to the abstraction layer and back enable transformations of semantically equivalent constraints of the same type between SHACL and these languages, possibly different in syntax and semantics. Hence, these mappings increase the interoperability of constraint languages with SHACL.

Default values for objects (*R-31*) or literals (*R-38*) of given properties within the context of given classes are inferred automatically when these properties are not present in the data. 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 for a certain book. The same semantically equivalent constraint of the type *default values* can be formulated in different languages:

```
SPIN:
owl:Thing
    spin:rule [ a sp:Construct ; sp:text """
        CONSTRUCT {
            ?subject :isPublished true . }
        WHERE {
            ?subject a :Book .
            FILTER NOT EXISTS { ?subject :isPublished ?literal } . } """ ; ] .

ReSh:
:BookResourceShape
    a oslc:ResourceShape ;
    oslc:property [
        oslc:propertyDefinition :isPublished ;
        oslc:defaultValue true ] .

SHACL:
:BookShape
    a sh:Shape ;
    sh:scopeClass :Book ;
    sh:property [
        sh:predicate :isPublished ;
        sh:defaultValue true ; ] .
```

Note that the SPIN representation of the constraint is a direct expression of the constraint using a SPIN rule which is executed for each resource of

the validation graph. A SPIN rule contains a SPARQL CONSTRUCT query which generates triples if all triple patterns within the SPARQL WHERE clause match.

By defining mappings from SHACL, ReSh, and SPIN constraints of the constraint type *default values* to the respective generic constraint (see Table 7.9) and back, arbitrary SHACL constraints of the constraint type *default values* can be converted to semantically equivalent constraints in SPIN and ReSh as well as the other way round.

Table 7.9. Generic Representation of the Constraint Type Default Values

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

7.4.3 Synchronizing Consistent Implementations of Constraint Types across Constraint Languages

Independently of the language used to express constraints, checks of semantically equivalent constraints should detect the same set of violations. As our framework is solely based on the abstract definitions of constraint types and as we provide just one SPIN mapping for each constraint type, it is ensured that the details of the SPIN implementation of a constraint type are consistent irrespective of the constraint language and that the validation of RDF data against semantically equivalent constraints of the same type in different languages leads always to exactly the same results.

By providing mappings from SHACL constraints of constraint types supported by SHACL to their generic intermediate representation, we are able to reuse once defined implementations for these constraint types that are consistent across constraint languages. By synchronizing the implementations for constraint types (1) we generically offer within our framework and (2) SHACL provides, constraint types remain consistently implemented.

7.5 Conclusion

We outlined our idea of a general framework to support the mapping of high-level constraint languages to an additional layer on top of SPARQL which can directly be used for validation by providing a mapping from the intermediate generic representation to SPIN which enables to actually validate RDF data against constraints provided in any RDF-based language.

We generalize from the experience completely implementing OWL 2 and DSP and introduce an abstraction layer that is able to describe constraints of any type in a way that mappings from high-level constraint languages to this intermediate generic representation can be created straight-forwardly. The additional abstraction layer reduces the representation of constraints to the absolute minimum that has to be provided in a mapping to SPIN.

The knowledge representation formalism Description Logics provides the foundational basis for the framework, which consists of a very simple conceptual model with a simple structure using a small lightweight vocabulary, the *RDF Constraints Vocabulary (RDF-CV)*. The core of the framework's building blocks comprises 103 constraining elements that are used to formulate constraints of all the 81 identified constraint types. In a technical report [51], 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 mappings using this framework.

We demonstrated how the framework can be used to map a constraint language to the RDF-CV and back. The latter enables the transformation of semantically equivalent constraints of the same type from one constraint language to another via the intermediate representation. This approach is also suitable to support the extension of constraint languages when additional constraint types should be supported by means of a simple bidirectional mapping.

Even though SPIN provides a convenient way to represent constraint violations and to validate RDF data, the implementation of a high-level constraint language still requires a tedious mapping to SPIN with a certain degree of freedom how constraints of a certain type are checked and how violations of constraints of a particular type are represented.

Our framework therefore provides a common ground that is solely based on the abstract definitions of constraint types. By providing just one SPIN mapping for each constraint type, it is ensured that the details of the SPIN implementation are consistent irrespective of the constraint language and that the validation of RDF data against semantically equivalent constraints of the same constraint type leads always to exactly the same results independently of the language used to express them.

Acknowledgements

Chapter 7 is based on 2 publications (1 article in the proceedings of a conference and 1 technical report). In a conference article, we give an overview of the suggested validation framework and its core building blocks, provide examples how constraints are generically represented using the additional layer on top of SPARQL, delineate how constraint types are consistently implemented across constraint languages, how constraint languages are mapped to the abstraction layer and back enabling transformations of semantically equivalent constraints expressed in different languages, and how the framework is used to simplify language implementations [45]. In a technical report, we list for each constraint type its representation in our framework, which not only shows that constraints of any type can indeed be described generically in this way, but also forms the starting point for any mappings using this framework [51].

8

The Role of Reasoning for RDF Validation

The set of constraint types forms the basis to investigate the role that reasoning and different semantics play 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. Validation and reasoning are closely related. Reasoning is beneficial for RDF validation as (1) it may resolve constraint violations, (2) it may cause valuable violations, and (3) it solves the redundancy problem.

A major shortcoming when validating RDF data is *redundancy*. Consider that a publication must have a publication date which is a typical constraint. When defining *books*, *conference proceedings*, and *journal articles* as sub-classes of *publication*, one would require to assign the concerned constraint explicitly to each sub-class, since instances of each of them should have a publication date. Reasoning is a promising solution as pre-validation step to overcome this shortcoming. *Reasoning* in Semantic Web refers to logical reasoning that makes implicitly available knowledge explicitly available. When performing reasoning one can infer that books must have a publication date from the facts that books are publications and publications must have a publication date. We remove redundancy by associating the constraint only with the super-class *publication*.

Users should be enabled to select on which constraint types to perform reasoning before data is validated and which constraint types to use to ensure data accuracy and completeness without reasoning. We therefore investigate the effect of reasoning to the validation process for each constraint type, i.e., we examine for each constraint type if reasoning may be performed prior to validation to enhance data quality either by resolving violations or by raising valuable violations and solving them (see Section 8.1).

We furthermore examine the effects of reasoning on the performance of constraint types. Hence, we investigate for each constraint type how efficient in terms of runtime validation is performed with and without reasoning. By mapping to *Description Logics* (*DL*) we get an idea of the performance for each constraint type in worst case, since the combination of DL constructs needed

to express a constraint type determines its computational complexity (see Section 8.1). For this reason, we determined which DL constructs are needed to express each constraint type (see Chapter E of the appendix [135]). Thus, the knowledge representation formalism DL, with its well-studied theoretical properties, provides the foundational basis for 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 (see Section 8.2).

Using these findings, we are able to determine which constraint types the six most common constraint languages enable to express (see Table 8.1 and Chapter B of the appendix [135] for more details on this evaluation). We use $\mathcal{CT}$ to refer to the whole set of constraint types, $\mathcal{R}$ to abbreviate the 35 constraint types for which reasoning may be performed before actually validating to enhance data quality and $\overline{\mathcal{R}}$ to denote the 46 constraint types for which reasoning does not improve data quality in any obvious sense.

Table 8.1. Expressivity of Constraint Languages regarding Reasoning

	$\mathcal{CT}$ (81)	$\overline{\mathcal{R}}$ (46)	$\mathcal{R}$ (35)
SPIN	**100.0** (81)	**100.0** (46)	**100.0** (35)
OWL 2 DL	**67.9** (55)	45.7 (21)	**97.1** (34)
SHACL	51.9 (42)	**52.2** (24)	51.4 (18)
ShEx	29.6 (24)	26.1 (12)	34.3 (12)
ReSh	25.9 (21)	15.2 (7)	40.0 (14)
OWL 2 QL	24.7 (20)	19.6 (9)	31.4 (11)
DSP	17.3 (14)	13.0 (6)	22.9 (8)

For OWL 2, we differentiate between the sub-languages OWL 2 QL and OWL 2 DL as they differ with regard to expressivity and efficiency in performance. Table 8.1 shows in percentage values (and absolute numbers in brackets) how many $\mathcal{CT}$, $\mathcal{R}$, and $\overline{\mathcal{R}}$ constraint types are supported by listed constraint languages. Although OWL 2 is the only language for which reasoning features are already implemented, $\mathcal{R}$ constraint types are also expressible by other languages.

Having information on the constraint type specific expressivity of constraint languages enables validation environments to recommend the right language depending on the users' individual use cases. These use cases determine which requirements have to be fulfilled and therefore which constraint types have to be expressed to meet these use cases.

The finding that SPIN is the only language that supports all reasoning constraint types underpins the importance to implement reasoning capabilities using SPIN (or plain SPARQL). The fact that all $\mathcal{R}$ and $\overline{\mathcal{R}}$ constraint

types are representable by SPIN emphasizes the significant role SPIN plays for the future development of constraint languages. Another important role may play OWL 2 DL with which 2/3 of all, nearly 1/2 of the $\overline{\mathcal{R}}$, and almost all $\mathcal{R}$ constraint types can be expressed. Even though some $\mathcal{R}$ constraint types correspond to OWL 2 DL axioms, we cannot use them directly to validate RDF data since OWL 2 reasoning and validation assume different semantics which may lead to differences in results. The upcoming W3C recommendation SHACL currently supports half of the $\mathcal{R}$ and half of the $\overline{\mathcal{R}}$ constraint types.

The contributions of this chapter are: (1) We work out 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. (2) For each constraint type, we examine if reasoning may improve data quality, how efficient in terms of runtime validation is performed with and without reasoning, and if validation results depend on the CWA and on the UNA. (3) We determine which reasoning constraint types the most common constraint languages enable to express and give directions for their further development. (4) We provide validation and reasoning implementations of constraint types[1] to be used to drive the further development of constraint languages (see Section 8.3).

8.1 Reasoning

DL provides the foundational basis for the expressive language *OWL 2* which offers knowledge representation and reasoning services. Validation is not the primary purpose of its design which has lead to claims that OWL 2 cannot be used for validation. In practice, however, OWL 2 is well-spread and RDFS/OWL 2 constructs are widely used to tell people and applications about how valid instances should look like. In general, RDF documents follow the syntactic structure and the semantics of RDFS/OWL 2 ontologies which could therefore not only be used for reasoning but also for validation.

In this section, we investigate the role that reasoning plays in practical data validation and how to overcome the major shortcomings when validating RDF data by performing reasoning prior to validation. As reasoning is beneficial for validation, we investigate the effect of reasoning to the validation process for each constraint type. Reasoning is beneficial for validation as (1) it may resolve constraint violations, (2) it may cause useful violations, and (3) it solves the redundancy problem. Consider the following DL knowledge base $\mathcal{K}$ - a DL knowledge base is a collection of formal statements which correspond to *facts* or *what is known* explicitly:

[1] Validation and reasoning implementations of constraint types on GitHub

$\mathcal{K}$ = {
Book ⊑ ∀ author.Person,
Book(Huckleberry-Finn),
author(Huckleberry-Finn, Mark-Twain),
Book ⊑ Publication,
Publication ⊑ ∃ publisher.Publisher,
editor ⊑ creator,
Book ⊑ ∀ identifier.ISBN }

As we know that books can only have persons as authors (Book ⊑ ∀ author.Person), *Huckleberry-Finn* is a book (Book(Huckleberry-Finn)), and *Mark Twain* is its author (author(Huckleberry-Finn, Mark-Twain)), we conclude that *Mark Twain* is a person. As *Mark Twain* is not explicitly defined to be a person, however, a violation is raised. Reasoning may resolve violations (1. benefit). If we apply reasoning before validating, the violation is resolved since the implicit triple Person(Mark-Twain) is inferred and therefore made explicitly available.

Reasoning may cause additional violations needed to enhance data quality in case these additional violations are resolved (2. benefit). As books are publications (Book ⊑ Publication), constraints on publications are also checked for books which may result in further valuable violations. As each publication must have a publisher (Publication ⊑ ∃ publisher.Publisher), e.g., a book is a publication, *Huckleberry-Finn* is a book, and *Huckleberry-Finn* does not have a publisher, a violation occurs. This violation would not have been raised without reasoning before actually validating and thus data quality would not be increased in case the violation is tackled.

The major shortcoming of classical constraint languages is redundancy. If a particular constraint should hold for multiple classes, it is required to assign the concerned constraint explicitly to each class. The redundancy problem is solved (3. benefit) by associating the constraint with the super-class of these classes and applying OWL 2 reasoning (see the introduction of this chapter).

Validation environments should enable users to select which constraint types to use for completing data by reasoning and which ones to consider as constraint types about data accuracy and completeness which could be checked over the data once completed using reasoning. As reasoning is beneficial for validating RDF data, we investigate the effect of reasoning to the validation process for each constraint type, i.e., we examine for each constraint type if reasoning may be performed prior to validation to enhance data quality either (1) by resolving violations or (2) by raising valuable violations. We denote the whole set of constraint types with $\mathcal{CT}$ which we divide into two disjoint sets:

1. $\mathcal{R}$ is the set of constraint types for which reasoning may be performed prior to validation (especially when not all the knowledge is explicit) to enhance data quality either by resolving violations or by raising valuable violations. For $\mathcal{R}$ constraint types, validation is executed by query answering with optional reasoning prior to validation. 35 (43.2%) of the overall 81 constraint types are $\mathcal{R}$ constraint types.

2. $\overline{\mathcal{R}}$ denotes the complement of $\mathcal{R}$, that is the set of constraint types for which reasoning cannot be done or for which reasoning does not improve data quality in any obvious sense. For $\overline{\mathcal{R}}$ constraint types, validation is performed by query answering without reasoning. 46 (56.8%) of the overall 81 constraint types are $\overline{\mathcal{R}}$ constraint types.

If a journal volume has an *editor* relationship to a person, then the journal volume must also have a *creator* relationship to the same person `(editor ⊑ creator)`,i.e., *editor* is a sub-property of *creator*. If we use *sub-properties* (*R-54/64*) without reasoning 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. If this triple is not present in the data, a violation occurs. If we use *sub-properties* with reasoning, however, the required triple is inferred which resolves the violation. *Sub-properties* is an $\mathcal{R}$ constraint type since reasoning may be performed prior to validation to improve data quality by resolving the violation. *Literal pattern matching* (*R-44*) restricts literals to match given patterns:

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

The first OWL 2 axiom explicitly declares *ISBN* to be a datatype. The second OWL 2 axiom defines *ISBN* as an abbreviation for a datatype restriction on *xsd:string*. The datatype *ISBN* can be used just like any other datatype such as in the universal restriction `Book ⊑ ∀ identifier.ISBN` which ensures that books can only have valid *ISBN* identifiers, i.e., strings that match a given regular expression. *Literal pattern matching* is an $\overline{\mathcal{R}}$ constraint type since reasoning cannot be done, i.e., does not change validation results and therefore does not improve data quality in any obvious sense.

For each constraint type we investigate how efficient in terms of runtime validation is performed with and without reasoning. By mapping to DL we get an idea of the performance of each constraint type in worst case, since the combination of DL constructs needed to express a constraint type determines its computational complexity.

8.1.1 Constraint Types with Reasoning

$\mathcal{R}$ is the set of constraint types for which reasoning may be performed prior to validation to enhance data quality either by resolving violations or by causing useful violations. For $\mathcal{R}$ constraint types, different types of reasoning may be performed which depends on the language used to formulate the constraint type. 11 of 35 $\mathcal{R}$ constraint types are representable by the less expressive but better performing OWL 2 QL. 23 $\mathcal{R}$ constraint types, in contrast, are not expressible by OWL 2 QL and therefore the more expressive but less performing OWL 2 DL is used. Some of the $\mathcal{R}$ constraint types, however, are also

representable by classical constraint languages (e.g., 40% are representable by ReSh).

OWL 2 profiles are restricted versions of OWL 2 that offer different trade-offs regarding expressivity vs. efficiency for reasoning. We consider the two extreme OWL 2 profiles - OWL 2 QL and OWL 2 DL - as OWL 2 QL is the profile with the highest performance and OWL 2 DL is the profile with the highest expressivity while still being a DL. *OWL 2 QL*, based on the *DL-Lite* family of DL [11, 75], is an OWL 2 profile which focuses on reasoning in the context of query answering with very large size of instance data. *OWL 2 DL* was standardized as a DL-like formalism with high expressivity, yet maintains decidability for main reasoning tasks. As a result of its expressive power, OWL 2 DL allows a large variety of sophisticated modeling capabilities for many application domains. The drawback of its expressive power results as a lack of computational efficiency in performance.

With regard to the two different types of reasoning we divide $\mathcal{R}$ into two not disjoint sets of constraint types: $\mathcal{R}_{QL} \subseteq \mathcal{R}_{DL}$ (OWL 2 DL is more expressive than OWL 2 QL).

1. $\mathcal{R}_{QL}$ is the set of $\mathcal{R}$ constraint types for which *OWL 2 QL reasoning* may be performed as they are expressible by OWL 2 QL. 11 of 35 $\mathcal{R}$ constraint types are $\mathcal{R}_{QL}$ constraint types.
2. $\mathcal{R}_{DL}$ stands for the set of $\mathcal{R}$ constraint types for which *OWL 2 DL reasoning* may be executed as OWL 2 QL is not expressive enough to represent them [227]. 34 of 35 $\mathcal{R}$ constraint types are $\mathcal{R}_{DL}$ constraint types.

Consider the following DL knowledge base $\mathcal{K}$ for the subsequent examples.

$\mathcal{K}$ = {
∃ author.⊤ ⊑ Publication,
author(Alices-Adventures-In-Wonderland, Lewis-Carroll),
Publication ⊑ ∃ publisher.Publisher,
publisher(A+Conference-Proceedings, A+Publisher),
rdf:type(A+Publisher, Publisher),
Publication ⊑ ∀ author.Person,
author(The-Lord-Of-The-Rings, Tolkien),
rdf:type(The-Lord-Of-The-Rings, Publication) }

OWL 2 QL Reasoning.

The *property domains* (*R-25*) constraint ∃ `author`.⊤ ⊑ `Publication` ensures that only publications can have *author* relationships (in OWL 2 QL: `author rdfs:domain Publication)`. Without reasoning, 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. With reasoning, on the contrary, the class assignment `rdf:type(Alices-Adventures-In-Wonderland, Publication)` is inferred which prevents the violation to be raised. Thus, reasoning improves data quality by resolving the violation. The *existential quantification* (*R-86*) `Publication` ⊑ ∃ `publisher.Publisher` restricts publications to have at least one publisher:

```
OWL 2 QL:
:Publication rdfs:subClassOf
  [ a owl:Restriction ;
    owl:onProperty :publisher ;
    owl:someValuesFrom :Publisher ] .
```

If reasoning is executed on the triples `publisher(A+Conference-Proceedings, A+Publisher)` and `rdf:type(A+Publisher, Publisher)`, it is inferred that *A+Conference-Proceedings* is a publication. Now, all constraints associated with publications are also validated for *A+Conference-Proceedings* - e.g., that publications must have at least one author. Without reasoning, in contrast, the fact that *A+Conference-Proceedings* is a publication is not explicit in the data which is the reason why constraints on publications are not validated for *A+Conference-Proceedings*. Hence, additional violations, which may be useful to enhance data quality in case the violations are taken into account, do not occur.

RDF validation with reasoning corresponds to performing SPARQL queries. As OWL 2 profiles are based on the *DL-Lite* family, OWL 2 QL is based on *DL-Lite*$_R$, and query answering in OWL 2 QL is performed in LogSpace (or rather in AC^0) [75], the same complexity class applies for validation by queries with reasoning. As TBox reasoning in OWL 2 QL is performed in Ptime [75], complete query rewriting as well as reasoning and subsequent querying (combined complexity) is carried out in Ptime [11, 75].

OWL 2 DL Reasoning.

Universal quantifications (*R-91*) are used to build anonymous classes containing all individuals that are connected by particular properties only to instances/literals of certain classes/data ranges. Publications, e.g., can only have persons as authors (`Publication` $\sqsubseteq \forall$ `author.Person`):

```
OWL 2 DL:
:Publication rdfs:subClassOf
  [ a owl:Restriction ;
    owl:onProperty :author ;
    owl:allValuesFrom :Person ] .
```

When performing reasoning, the triples `author(The-Lord-Of-The-Rings, Tolkien)` and `rdf:type(The-Lord-Of-The-Rings, Publication)` let a reasoner infer that *Tolkien* is a person which satisfies the *universal quantification*. In case reasoning is not executed, a violation is raised since it is not explicitly stated that *Tolkien* is a person. As a consequence, constraints on persons are not checked for *Tolkien* which prevents further validation.

With OWL 2 DL, the more expressive profile than OWL 2 QL, reasoning is executed in N2Exptime [227] which is a class of considerably higher complexity than Ptime, the complexity class for OWL 2 QL reasoning. As we consider ontological reasoning, complexity classes are assigned to sets of constraint types according to well-established complexity results in literature on

reasoning of DL languages. Therefore, the classification also includes complex logical interferences between TBox axioms.

8.1.2 Constraint Types without Reasoning

$\overline{\mathcal{R}}$ is the set of constraint types for which reasoning cannot be done or for which reasoning does not improve data quality in any obvious sense. *Context-specific exclusive or of properties* (*R-11*) is a $\overline{\mathcal{R}}$ constraint type with which it can be defined that an individual of a certain class can either have property A or property B, but not both. Identifiers of publications, e.g., can either be ISBNs (for books) or ISSNs (for periodical publications), but it should not be possible to assign both identifiers to a given publication:

$$\begin{aligned}\mathcal{K} = \{ & \\ & \texttt{Publication} \sqsubseteq (\neg \texttt{A} \sqcap \texttt{B}) \sqcup (\texttt{A} \sqcap \neg \texttt{B}), \\ & \texttt{A} \equiv \exists\ \texttt{isbn.xsd:string}, \\ & \texttt{B} \equiv \exists\ \texttt{issn.xsd:string}\ \}\end{aligned}$$

This constraint can be represented by OWL 2 DL by building an anonymous class for each exclusive property:

```
OWL 2 DL:
:Publication owl:disjointUnionOf ( :A :B ) .
:A rdfs:subClassOf [ a owl:Restriction ;
    owl:onProperty :isbn ;
    owl:someValuesFrom xsd:string ] .
:B rdfs:subClassOf [ a owl:Restriction ;
    owl:onProperty :issn ;
    owl:someValuesFrom xsd:string ] .
```

Exactly the same constraint can be expressed by ShEx more intuitively and concisely:

```
ShEx:
:Publication  ( :isbn xsd:string | :issn xsd:string )
```

It is a common requirement to narrow down the value space of properties by an exhaustive enumeration of valid values (*R-30/37: allowed values*). Reasoning on this constraint type does not change validation results and therefore does not improve data quality. Books on the topics *Computer Science* and *Librarianship*, e.g., should only have *ComputerScience* and *Librarianship* as subjects. The corresponding DL statement `Book` $\equiv \forall$ `subject.{ComputerScience,Librarianship}` is representable by DSP and OWL 2 DL:

```
DSP:
[ a dsp:DescriptionTemplate ;
  dsp:resourceClass :Book ;
  dsp:statementTemplate [
    dsp:property :subject ;
    dsp:nonLiteralConstraint [
      dsp:valueURI :ComputerScience, :Librarianship ] ] ] .

OWL 2 DL:
:subject rdfs:range [ owl:equivalentClass [ a owl:Class;
    owl:oneOf ( :ComputerScience :Librarianship ) ] ] .
```

RDF validation without reasoning corresponds to performing SPARQL queries. It is known that performing SPARQL queries is carried out in PSPACE-Complete [255]. Table 8.2 gives an overview over the complexity of validation with and without reasoning. The higher the complexity class the worse the performance. The order of the complexity classes is the following [10]:

$$\text{LOGSPACE} \subseteq \text{PTIME} \subseteq \text{PSPACE-Complete} \subseteq \text{N2EXPTIME}$$

Table 8.2. Complexity of Validation with and without Reasoning

Validation Type	Complexity Class
$\overline{\mathcal{R}}$	PSPACE-Complete
$\mathcal{R}_{QL}$	PTIME
$\mathcal{R}_{DL}$	N2EXPTIME

We do not consider *OWL 2 Full* due to its high worst case complexity (undecidability) and as all (except of one) $\mathcal{R}$ constraint types are already expressible either by OWL 2 QL or OWL 2 DL.

8.2 CWA and UNA Dependency

RDF validation and OWL reasoning assume different semantics. Reasoning in OWL 2 is based on the *open-world assumption* (*OWA*), i.e., a statement cannot be inferred to be false if it cannot be proved to be true which fits its primary design purpose to represent knowledge on the World Wide Web. Consider the following DL knowledge base $\mathcal{K}$:

$\mathcal{K} = \{$
`Book` $\sqsubseteq \exists$ `title.`$\top$,
`Book(Hamlet)`,
`Publication` $\sqsubseteq \geqslant$`1 author.Person`,
`Book` $\sqcap$ `JournalArticle` $\sqsubseteq \bot$,
`funct (title)`,
`title(Huckleberry-Finn, The-Adventures-of-Huckleberry-Finn)`,
`title(Huckleberry-Finn, Die-Abenteuer-des-Huckleberry-Finn)` $\}$

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

OWL 2 is based on the *non-unique name assumption* (*nUNA*) whereas RDF validation requires that different names represent different objects (*unique name assumption* (*UNA*)). Although DLs/OWL 2 do not assume the UNA, they have the constructs *owl:sameAs* and *owl:differentFrom* to state that two names are the same or different. If validation would assume the OWA and the nUNA, validation won't be that restrictive and therefore we won't get the intended validation results. This ambiguity in semantics is one of the main reasons why OWL 2 has not been adopted as the standard constraint language for RDF validation in the past.

RDF validation and reasoning assume different semantics which may lead to different validation results when applied to particular constraint types. Hence, 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 the constraint type depends on the CWA and (2) if the constraint type depends on the UNA.

We classify constraint types according to the dependency on the CWA and the UNA which leads to four sets of constraint types: (1) $\mathcal{CWA}$ denotes the set of constraint types which are dependent on the CWA, i.e., the set of constraint types for which it makes a difference in terms of validation results if the CWA or the OWA is assumed. *Minimum qualified cardinality restrictions* (*R-75*) is a $\mathcal{CWA}$ constraint type. Publications, e.g., must have at least one author $\texttt{(Publication} \sqsubseteq \geqslant \texttt{1 author.Person)}$. In a CWA setting, a publication without an explicitly stated author violates the constraint, whereas with OWA semantics, a publication without an explicitly stated author does not raise a violation as the constraint entails that there must be an author which we may not know.

(2) $\overline{\mathcal{CWA}}$ is the complement of $\mathcal{CWA}$ and thus includes constraint types which are independent on the CWA. Nothing can be a book and a journal article at the same time $\texttt{(Book} \sqcap \texttt{JournalArticle} \sqsubseteq \bot\texttt{)}$. For the constraint type *disjoint classes* (*R-7*), it does not make any difference regarding validation results if the CWA or the OWA is taken, as if there is a publication which is a book and a journal article a violation is raised in both settings, i.e., additional information does not change validation results.

(3) $\mathcal{UNA}$ denotes the set of constraint types which are dependent on the UNA. For *functional properties* (*R-57/65*), it makes a difference with regard to validation results if the UNA or the nUNA is assumed. As the property *title* is functional `(funct (title))`, a book can have at most one distinct title. UNA causes a clash if the book *Huckleberry-Finn* has more than one title. For nUNA, however, reasoning concludes that the title *The-Adventures-of-Huckleberry-Finn* must be the same as the title *Die-Abenteuer-des-Huckleberry-Finn* which resolves the violation.

(4) $\overline{\mathcal{UNA}}$, the complement of $\mathcal{UNA}$, denotes the set of constraint types which are independent on the UNA. *Literal value comparison* (*R-43*) is an example of a $\overline{\mathcal{UNA}}$ constraint type which ensures that, depending on property datatypes, two different literal values have a specific ordering with re-

spect to an operator like <, <=, >, and >=. It has to be guaranteed, e.g., that birth dates 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. The *literal value comparison* constraint type is independent from the UNA as the violation is not resolved in case there are further resources (e.g., *AlbertEinstein*, *Albert_Einstein*) which point to correct birth and death dates and which may be the same as the violating resource when nUNA is assumed.

We evaluated for each constraint type if it is dependent on the CWA and the UNA (for the detailed analysis we refer to Chapter D of the appendix [135]). The result is that we distinguish between 46 (56.8%) $\mathcal{CWA}$ and 35 (43.2%) $\overline{\mathcal{CWA}}$ and between 54 (66.6%) $\mathcal{UNA}$ and 27 (33.3%) $\overline{\mathcal{UNA}}$ constraint types. Hence, for the majority of the constraint types it makes a difference in terms of validation results if the CWA or the OWA and if the UNA or the nUNA is assumed. For the $\mathcal{CWA}$ and the $\mathcal{UNA}$ constraint types, we have to be careful in case we want to use them for reasoning and for validation as in both usage scenarios we assume different semantics which leads to different results.

8.3 Implementation

We use *SPIN*, a SPARQL-based way to formulate and check constraints, as basis to develop a validation environment, online available at http://purl.org/net/rdfval-demo, to validate RDF data according to constraints expressed by arbitrary constraint languages by mapping them to SPIN [44]. The SPIN engine checks for each resource if it satisfies all constraints, which are associated with its classes, and generates a result RDF graph containing information about all constraint violations. We provide implementations for all constraint types expressible by OWL 2 QL, OWL 2 DL, and DSP as well as for major constraint types representable by ReSh and ShEx.[2]

By means of a *property ranges* (*R-35*) constraint it can be restricted that *author* relations can only point to persons (DL: $\top \sqsubseteq \forall$ `author.Person`, OWL 2 DL: `author rdfs:range Person.`). There is one SPIN construct template for each constraint type, so for the constraint type *property ranges*:

```
owl2spin:PropertyRanges a spin:ConstructTemplate ;
  spin:body [ a sp:Construct ; sp:text """
    CONSTRUCT {
      _:constraintViolation a spin:ConstraintViolation [...] . }
    WHERE {
      ?OP rdfs:range ?C . ?x ?OP ?subject . ?subject a owl:Thing .
      FILTER NOT EXISTS { ?subject a ?C } . } """ ; ] .
```

[2] Implementations of constraint types on GitHub

A SPIN construct template contains a SPARQL CONSTRUCT query generating constraint violation triples which indicate the subject, the properties, and the constraint causing the violations and the reason why violations have been raised. Violation triples, which are associated with a certain level of severity (informational, warning, error), may also give some guidance how to fix them. A SPIN construct template creates violation triples if all triple patterns within the SPARQL WHERE clause match. If *Doyle*, the author of the book *Sherlock-Holmes* `(author(Sherlock-Holmes, Doyle)),` e.g., is not explicitly declared to be a person, all triple patterns within the SPARQL WHERE clause match and the SPIN construct template generates a violation triple.

Property ranges is an $\mathcal{R}$ constraint type, i.e., a constraint type for which reasoning may be performed prior to validation to enhance data quality. Therefore, validation environments should enable users to decide if reasoning on *property ranges* constraints should be executed before validation. If a user decides to use reasoning, the triple `rdf:type(Doyle, Person),` whose absence caused the violation, is inferred before data is validated which resolves the violation.

Validation environments should enable users (1) to select individual resources for which reasoning should be performed on $\mathcal{R}$ constraints before they are validated, (2) to select $\mathcal{R}$ constraint types for which reasoning should be executed, and (3) to globally determine if for all $\mathcal{R}$ constraint types reasoning should be done. All resources, for which reasoning should be performed prior to validation, are automatically assigned to the class *Reasoning* during a pre-reasoning step. There is one SPIN rule for each $\mathcal{R}$ constraint type, so for *property ranges*:

```
owl2spin:Reasoning spin:rule [ a sp:Construct ; sp:text """
  CONSTRUCT { ?subject a ?C . }
  WHERE { ?OP rdfs:range ?C . ?x ?OP ?subject . ?subject a owl:Thing .
    FILTER NOT EXISTS { ?subject a ?C } . } """ ; ] .
```

The SPIN rule is executed for each resource of the class *Reasoning*. A SPIN rule contains a SPARQL CONSTRUCT query which generates triples if all triple patterns within the SPARQL WHERE clause match. In case *Doyle* is not defined to be a person, all triple patterns match and the triple `rdf:type(Doyle, Person)` is created. As a consequence, the violation on *Doyle* is not raised. We implemented reasoning capabilities for all $\mathcal{R}$ constraint types for which OWL 2 QL and OWL 2 DL reasoning may be performed.[3]

8.4 Conclusion

We clearly showed that for the majority of the constraint types validation results differ depending on whether validation is based on the CWA or the OWA and whether the UNA or the nUNA is underlying. For these constraint

[3] Implementation of reasoning capabilities for reasoning constraint types on GitHub

types, we have to be careful in case we want to use them for reasoning and for validation as in both usage scenarios we assume different semantics which leads to different results.

Equally, using or not using reasoning has serious impact on which constraints are considered to be violated or fulfilled. Obviously, these findings are not new and should be clear to everyone working with RDF and Semantic Web technologies. According to our experience, however, the topic data validation is far too often reduced to the selection of suitable constraint languages which may be related to the obvious but yet inaccurate comparison of RDF with XML as basis technology to represent data.

With this chapter, we want to make clear that it depends on more than just the constraint language when validating RDF data and when developing appropriate systems. Therefore, in order to realize interoperable solutions for data validation, three components are needed: (1) An adequate constraint language is required that allows to represent the desired constraints. (2) The underlying semantics has to be specified, be it open or closed world, particularly if constraint types are used that depend on the choice of the semantics. (3) It must be determined if reasoning should be involved in the validation process or not. Necessary reasoning steps have to be predefined to allow the correct interpretation of constraints, e.g., when constraints are defined on super-classes to avoid redundancy.

Users should be enabled to select on which constraint types reasoning should be performed before data is validated and which constraint types to use to ensure data accuracy and completeness without reasoning. We therefore investigated for each constraint type if reasoning may be performed prior to validation to enhance data quality either by resolving violations or by raising valuable violations. For more than 2/5 of the constraint types, reasoning may be performed to improve data quality. For less than 3/5 of the constraint types, however, reasoning cannot be done or does not improve data quality in any obvious sense.

For reasoning constraint types, different types of reasoning may be performed which depends on the language used to formulate the constraint type. 1/3 of the reasoning constraint types are representable by the less expressive but better performing OWL 2 profile *OWL 2 QL* and thus *OWL 2 QL reasoning* may be performed. The remaining reasoning constraint types, in contrast, are not expressible by OWL 2 QL and therefore the more expressive but less performing OWL 2 sub-language *OWL 2 DL* is used.

For each constraint type, we examined how efficient in terms of runtime validation is performed with and without reasoning in worst case. By mapping to Description Logics we get an idea of the performance for each constraint type in worst case, since the combination of Description Logics constructs needed to express a constraint type determines its computational complexity.

Acknowledgements

Chapter 8 is based on 2 publications (1 article in the proceedings of a conference and 1 technical report) to investigate the role that reasoning and different semantics play in practical data validation. For one article, we examined for each constraint type (1) if reasoning may be performed prior to validation 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. Using these findings, we determined for the most common constraint languages which constraint types they enable to express and give directions for the further development of constraint languages [39]. In a technical report, we exhaustively delineate the whole investigation for each constraint type [51].

9

Evaluating the Usability of Constraint Types for Assessing RDF Data Quality

Based on our work in the DCMI and the W3C working groups and the requirements jointly identified within these working groups, we published by today 81 types of constraints that are required by various stakeholders for data applications. A concrete constraint is instantiated from one of the 81 constraint types and defined for a specific vocabulary.

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 practitioners and therefore enables to express each of the 81 identified constraint types. This is the reason why further research and development in the field of constraint languages is needed.

In order to gain a better understanding about the role of certain constraint types for determining the quality of RDF data and thus to evaluate the usability of identified constraint types for assessing RDF data quality, we (1) collect 115 constraints of different types for three vocabularies commonly used in the social, behavioral, and economic (SBE) sciences, either from the vocabularies themselves or from several domain experts, (2) classify these constraints, and (3) conduct a large-scale evaluation by validating data against these constraints.

These vocabularies are representative, cover different aspects of SBE research data, and are a mixture of widely adopted, accepted, and well-established vocabularies (QB and SKOS) and a vocabulary under development (DDI-RDF).

We classify each constraint based on which types of constraint languages are able to express its constraint type. Furthermore, we let the domain experts classify the constraints according to the severity of their violation. Although we provide default severity levels for each constraint, validation environments should enable users to adapt the severity levels of constraints according to their individual needs (see Section 9.1).

As we do not want to base our conclusions on the evaluation of vocabularies, constraint definitions, and constraint classifications alone, we conduct a

large-scale experiment. We evaluate the data quality of 15,694 data sets (4.26 billion triples) of SBE research data, obtained from 33 SPARQL endpoints, by validating against these 115 constraints.

Based on the evaluation results, we formulate several findings to gain valuable insights for future developments in this field and make recommendations to direct the further development of constraint languages. To make valid general statements for all vocabularies, however, these findings still have to be verified or falsified by evaluating the quality of RDF data represented by more than three vocabularies (see Section 9.2).

We discuss constraints on RDF data in general. Note that the data represented in RDF can be data in the sense of the SBE sciences, but also metadata about published or unpublished data. We generally refer to both simply as RDF data.

9.1 Classification of Constraint Types and Constraints

To gain insights into the role that certain types of constraints play for assessing the quality of RDF data, we use two simple classifications. On the one hand, we classify constraint types based on whether they are expressible by different types of constraint languages (see Chapter B of the appendix [135] for the evaluation of the constraint type specific expressivity of the six most common constraint languages). On the other hand, we let several domain experts classify constraints, which are formulated for a given vocabulary, according to the perceived severity of their violation.

For the three vocabularies, several SBE domain experts determined the severity level for each of the 115 constraints. In a technical report, we provide detailed textual descriptions for all these constraints as well as their assignments to constraint types and severity levels [137]. In the following, we summarize the classifications of constraint types and constraints for the purpose of our evaluation.

9.1.1 Classification of Constraint Types according to the Expressivity of Constraint Language Types

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

1. *RDFS/OWL Based*
2. *Constraint Language Based*
3. *SPARQL Based*

RDFS/OWL Based. *RDFS/OWL Based* denotes the set of constraint types which can be formulated with RDFS/OWL axioms when using them in terms of constraints with CWA/UNA semantics and without reasoning.[1]

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

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

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

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

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

[1] The entailment regime is to be decided by the implementers. It is our point that reasoning affects validation and that a proper definition of the reasoning to be applied is needed.

Context-specific exclusive or of property groups (*R-13*) is a constraint type which can be formulated by the high-level constraint language ShEx. Constraints of this type restrict individuals of given classes to have property links of properties defined within exactly one of multiple mutually exclusive property groups. Within the context of DDI-RDF, e.g., *skos:Concept*s can have either *skos:definition* (when interpreted as theoretical concepts) or *skos:notation* and *skos:prefLabel* properties (when interpreted as codes), but not both:

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

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

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

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

9.1.2 Classification of Constraints according to the Severity of Constraint Violations

A concrete constraint is instantiated from one of the 81 constraint types and defined for a specific vocabulary. It does not make sense to determine the severity of constraint violations of an entire constraint type, as the severity depends on the individual context and vocabulary. SBE experts determined the default *severity level*[2] for each of the 115 constraints to indicate how serious the violation of the constraint is. Although we provide default severity levels for each constraint, validation environments should enable users to adapt the severity levels of constraints according to their individual needs.

[2] The possibility to define severity levels for concrete constraints is in itself a requirement (*R-158*).

We use the classification system of log messages in software development like *Apache Log4j 2* [7], the *Java Logging API*,[3] and the *Apache Commons Logging API*[4] as many data practitioners also have experience in software development and software developers intuitively understand these levels. We simplify this commonly accepted classification system and distinguish the three severity levels (1) *informational*, (2) *warning*, and (3) *error*. Violations of *informational* constraints point to desirable but not necessary data improvements to achieve RDF representations which are ideal in terms of syntax and semantics of used vocabularies. *Warnings* indicate syntactic or semantic problems which typically should not lead to an abortion of data processing. *Errors*, in contrast, are syntactic or semantic errors which should cause the abortion of data processing.

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

9.1.3 Classification Examples

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

RDFS/OWL Based. It is a common requirement to narrow down the value space of properties by an exhaustive enumeration of valid values (*R-30/37: allowed values*): *disco:CategoryStatistics*, e.g., can only have *disco:computationBase* relationships to the literals *"valid"* and *"invalid"* of the datatype *rdf:langString* (default severity level: *error*). Consider the following *DL knowledge base* $\mathcal{K}$:[5]

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

The constraint type *existential quantifications* (*R-86*) can be used to enforce that instances of given classes must have some property relation to individuals/literals of certain types. Variables, e.g., should have a relation to

[3] http://docs.oracle.com/javase/7/docs/api/java/util/logging/Level.html
[4] http://commons.apache.org/proper/commons-logging
[5] A *DL knowledge base* is a collection of formal statements which correspond to *facts* or what is known explicitly.

a theoretical concept (*informational*). The default severity level of this constraint is weak, as in most cases research can be continued without having information about the theoretical concept of a variable.

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

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

It is often useful to declare a given (data) property as the *primary key* (*R-226*) of a class, so that a system can enforce uniqueness and build URIs from user inputs and imported data. In DDI-RDF, resources are uniquely identified by the property *adms:identifier*, which is therefore inverse-functional (`funct identifier`$^-$), i.e., for each *rdfs:Resource* x, there can be at most one distinct resource y such that y is connected by *adms:identifier*$^-$ to x (*error*). Keys, however, are even more general than *inverse-functional properties* (*R-58*), as a key can be a data property, an object property, or a chain of properties [279]. For this reason, as there are different sorts of key, and as keys can lead to undecidability, DL is extended with the construct *keyfor* (`identifier keyfor Resource`) [211] which is implemented by the OWL 2 *hasKey* construct.

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

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

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

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

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

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

For data properties, it may be desirable to restrict that values of predefined languages must be present for determined number of times (*R-48/49: language tag cardinality*): (1) It is checked if literal language tags are set. Some controlled vocabularies, e.g., contain literals in natural language, but without information what language has actually been used (*warning*). (2) Language tags must conform to language standards (*error*). (3) Some thesaurus concepts are labeled in only one, others in multiple languages. It may be desirable to have each concept labeled in each of the languages that are also used on the other concepts, as language coverage incompleteness for some concepts may indicate shortcomings of thesauri (*informational*) [213].[6]

9.2 Evaluation

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

Despite the large volume of the evaluated data sets in general, we have to keep in mind that in this study we only validate data against constraints for three vocabularies. We took all well-established and newly developed SBE vocabularies into account and defined constraints for the three vocabularies of them, which are most commonly used in the SBE sciences. For these three vocabularies, large data sets have already been published. For other SBE vocabularies, however, there is often not (yet) enough data openly available to draw general conclusions. Yet, these three vocabularies are representative, cover different aspects of SBE research data, and are also a mixture of widely adopted, accepted, and well-established vocabularies (QB and SKOS) on the one side and a vocabulary under development (DDI-RDF) on the other side.

[6] [213] investigated how to support taxonomists in improving SKOS vocabularies by pointing out quality issues that go beyond the integrity constraints defined in the SKOS specification.

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

9.2.1 Experimental Setup

On the three vocabularies DDI-RDF, QB, and SKOS, we collected 115 constraints,[7] either from the vocabularies themselves or from several domain experts, and classified and implemented them for data validation. We ensured that these constraints are equally distributed over the sets and vocabularies we have. We then evaluated the data quality of 15,694 data sets (4.26 billion triples) of SBE research data, obtained from 33 SPARQL endpoints,[8] by validating the data against these 115 constraints. Table 9.1 lists the number of validated data sets and the overall sizes in terms of validated triples for each of the vocabularies.

Table 9.1. Number of Validated Data Sets and Triples for each Vocabulary

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

We validated, i.a., (1) QB data sets published by the *Australian Bureau of Statistics*,[9] the *European Central Bank*,[10] and the *Organisation for Economic Co-operation and Development*,[11] (2) SKOS thesauri like the *AGROVOC Multilingual agricultural thesaurus*,[12] the *STW Thesaurus for Economics*,[13] and the *Thesaurus for the Social Sciences*,[14] and (3) DDI-RDF data sets provided by the *Microdata Information System*,[15] the *Data Without Boundaries Dis-*

[7] Implementations of all 115 constraints on GitHub

[8] To get further information about how to access individual SPARQL endpoints, we refer to a technical report in which we describe the complete evaluation in further detail [138].

[9] http://abs.gov.au

[10] http://www.ecb.europa.eu/home/html/index.en.html

[11] http://www.oecd.org

[12] http://202.45.139.84:10035/catalogs/fao/repositories/agrovoc

[13] http://zbw.eu/stw/versions/latest/about

[14] http://www.gesis.org/en/services/research/thesauri-und-klassifikationen/social-science-thesaurus

[15] http://www.gesis.org/missy

covery Portal,[16] the *Danish Data Archive*,[17] and the *Swedish National Data Service*.[18] In a technical report, we describe the evaluation in further detail for each vocabulary, queried SPARQL endpoint, and constraint [138]. Furthermore, we have published the evaluation results[19] for each QB data set in form of one document per SPARQL endpoint.

Since the validation of each of the 81 constraint types can be implemented using SPARQL, we use SPIN, a SPARQL-based way to formulate and check constraints, as basis to develop a validation environment to validate RDF data according to constraints expressed by arbitrary constraint languages. The validation environment can directly be used to validate arbitrary RDF data according to constraints extracted from the three vocabularies.[20] Additionally, own constraints on any vocabulary can be defined using several constraint languages.

The SPIN engine checks for each resource if it satisfies all constraints, which are associated with its assigned classes, and generates a result RDF graph containing information about all constraint violations. There is one SPIN construct template for each constraint type. A SPIN construct template contains a SPARQL CONSTRUCT query which generates constraint violation triples indicating the subject and the properties causing constraint violations and the reason why constraint violations have been raised. A SPIN construct template creates constraint violation triples if all triple patterns within the SPARQL WHERE clause match.

9.2.2 Evaluation Results and Formulation of Findings

Tables 9.2 and 9.3 show the results of the evaluation, more specifically the numbers of constraints and constraint violations, which are caused by these constraints, in percent; whereas the numbers in the first line indicate the absolute amount of constraints and violations. The constraints and their raised violations are grouped by vocabulary, which type of language the types of the constraints are formulated with, and their severity level.

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

There is a strong overlap between *RDFS/OWL* and *Constraint Language Based* constraint types as in many cases constraint types are expressible by RDFS/OWL and classical constraint languages. This is the reason why the

[16] http://dwb-dev.nsd.uib.no/portal
[17] https://www.sa.dk/en/services/danish-data-archive
[18] http://snd.gu.se/en
[19] Evaluation results on GitHub
[20] Vocabulary implementations on GitHub

percentage values of constraints and violations grouped by the classification of constraint types according to the expressivity of constraint language types do not accumulate to 100%.

Table 9.2. Evaluation Results (1)

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

C (constraints), CV (constraint violations)

Table 9.3. Evaluation Results (2)

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

C (constraints), CV (constraint violations)

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

Finding 1 *The facts that 80% of all violations are raised by SPARQL Based constraints and that 2/3 of all constraints are SPARQL Based, increases the importance to formulate constraints, which up to now can only be expressed in SPARQL, using high-level constraint languages. Data quality can be significantly improved when suitable constraint languages are developed which enable to define SPARQL Based constraints in an easy, concise, and intuitive way. Thereby, the more elaborate a vocabulary is, the more sophisticated and complex constraints are specified using SPARQL.*

These constraints are of such complexity that up to now in most cases they can only be expressed by plain SPARQL. It should be an incentive for language designers to devise languages which are more intuitive than SPARQL

in a way that also domain experts, which are not familiar with SPARQL, can formulate respective constraints.

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

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

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

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

For DDI-RDF, data providers still have to understand the vocabulary and of course data cannot have high quality if the specification is not yet stable. It is likely that a newly developed vocabulary is still subject of constant change and that early adopters did not properly understand its formal specification. Thus, published data may not be consistent with the current draft of its conforming vocabulary.

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

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

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

Finding 5 *Although 40% of all constraints are error constraints, the percentage of severe violations is very low, compared to about 2/3 of warning and 1/3*

of informational violations. This implies that data quality is high with regard to the severity level of constraints and that proper constraint languages can significantly improve data quality beyond fundamental requirements.

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

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

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

80% of the violations, which are raised by either *RDFS/OWL* or *Constraint Language Based* constraints, are caused by constraints with the severity level *informational* (see Table 9.4) and almost all (94%) of the violations, which are caused by *SPARQL Based* constraints, are raised by *warning* constraints. Approximately 1/2 of all constraints are *informational* constraints regardless how their types are classified according to the expressivity of constraint language types.

Table 9.4. Constraints and Violations by Language Type and Severity

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

C (constraints), CV (constraint violations)

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

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

Finding 8 *Regardless of the type of the used language, there are only a few violations raised by error constraints which stands for good data quality in*

general. In contrast, constraints of low severity, expressible by RDFS/OWL or high-level constraint languages, are violated to a large extent (80%), whereas more serious warning constraints, only expressible by SPARQL, are violated to an even larger extend (94%).

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

9.3 Conclusion

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

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

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

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

1. *Data quality can be significantly improved when suitable constraint languages are developed enabling to define constraints - which up to now can only be expressed by plain SPARQL - in an easy, concise, and intuitive way. Thereby, the more elaborate a vocabulary is, the more sophisticated and complex constraints are necessary, which in most cases can only be specified in SPARQL.*
2. *As only 1/5 of all violations result from constraints which are expressible in RDFS or OWL, even though 1/3 of all constraints are representable in*

RDFS/OWL, data quality is high for all vocabularies with regard to their formal specifications.

3. *Although 40% of all constraints are associated with the severity level error, the percentage of severe violations is very low - compared to about 2/3 of warning and 1/3 of informational violations. This implies that data quality is high with regard to the severity of constraints and that proper constraint languages can significantly improve data quality beyond fundamental requirements.*
4. *Whatever language is used to formulate constraints, 1/2 of all constraints are informational, 1/3 are error, and 1/5 are warning constraints. This fact emphasizes the importance that constraint languages support multiple levels of severity.*
5. *Violations caused by constraints representable by RDFS/OWL or high-level constraint languages are of low severity, whereas the violation of constraints, only expressible in SPARQL, is more serious. This is the reason why there is a significant demand for high-level constraint languages that support the expression of constraints which up to now can only be formulated by plain SPARQL.*

Acknowledgements

Chapter 9 is based on 4 publications: 1 journal article, 1 article in the proceedings of a conference, and 2 technical reports. We (1) collected and classified 115 constraints for three vocabularies commonly used in the SBE sciences, (2) evaluated the data quality of 15,694 data sets (4.26 billion triples) of SBE research data, obtained from 33 SPARQL endpoints, by validating the data against these constraints, and (3) formulated several findings to direct the further development of constraint languages based on the evaluation results [139, 140]. A technical report serves to describe each of the 115 constraints, assign them to constraint types, and classify them according to the severity of their violation [137]. In a second technical report, we meticulously delineate the conducted large-scale evaluation for each vocabulary, queried SPARQL endpoint, and constraint [138].

10

Discussion

In this chapter, we summarize the main contributions and limitations of this thesis based on how the research questions are answered, present an outlook on upcoming research directions, and finally conclude this work.

10.1 Contributions and Limitations

Research Question 1 *Which types of research data and related metadata are not yet representable in RDF and how to adequately model them to be able to validate RDF data against constraints extractable from these vocabularies?*

The *social, behavioral, and economic (SBE) sciences* require high-quality data for their empirical research. There are different types of research data and related metadata. We define each of these types as stated by data professionals, delineate their transition in the research data lifecycle, and show which RDF vocabularies are commonly used to represent them.

Contribution 1 *Development of three RDF vocabularies (1) to represent all types of research data and related metadata in RDF and (2) to validate RDF data against constraints extractable from these vocabularies*

Because of the lack of appropriate vocabularies, however, just a few of these types are expressible in RDF. We have developed three vocabularies (1) to represent all types of research data and its metadata in RDF and (2) to validate RDF data according to constraints extractable from these vocabularies (see Chapter 3):

- The *DDI-RDF Discovery Vocabulary (DDI-RDF)* [42] supports the discovery of metadata on *unit-record data*, the type of data most often used in research within the SBE sciences, i.e., data collected about individuals, businesses, and households. It can be applied to research data from many different domains, rather than being specific to a single set of domain data. DDI-RDF is based on a metadata standard, composed of almost

twelve hundred metadata fields, known as the *Data Documentation Initiative (DDI)* [95], an XML format to disseminate, manage, and reuse data collected and archived for research.
- *Physical Data Description (PHDD)* [321] is a vocabulary to describe data in tabular format and its physical properties. The data could either be represented in form of records with character-separated values (CSV) or with fixed length.
- The *SKOS Extension for Statistics (XKOS)* [84] is a vocabulary to describe the structure and textual properties of formal statistical classifications as well as relations between classifications and concepts, and to introduce refinements of SKOS semantic properties to allow the use of more specific relations between concepts.

Research Question 2 *How to directly validate XML data on semantically rich OWL axioms using common RDF validation tools when XML Schemas, adequately representing particular domains, have already been designed?*

Data practitioners 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 properly describing conceptual models about data of certain domains, we propose on formal logics and the XML Schema (XSD) meta-model based automatic transformations of arbitrary XSDs and conforming XML documents into OWL ontologies and corresponding RDF data (see Chapter 4). This does not cause any additional manual effort as these constraints have already been defined within underlying XSDs. Semantically rich OWL axioms against which XML data can directly be validated are (1) subclass relationships, (2) OWL hasValue restrictions on data properties, and (3) OWL universal restrictions on object properties.

Contribution 2 *Direct validation of XML data using common RDF validation tools against semantically rich OWL axioms extracted from XML Schemas properly describing certain domains*

OWL axioms are extracted out of XSDs either (1) explicitly according to XSD constructs determining the syntactic structure of sets of XML documents or (2) implicitly according to the implicit semantics of XSD constructs. To model explicit and implicit semantics in a semantically correct way, we formally underpin transformations themselves on semantically rich OWL axioms for which Description Logics with its well-studied theoretical properties provides the foundational basis.

As structures of XSDs may be quite complex, we base these transformations on the XSD meta-model and map each of its constructs to suitable axioms of an OWL TBox. This enables to translate arbitrary complex struc-

tures of XSDs into OWL using identical transformation rules and therefore ensures that any XSD can be converted without any information loss.

As XSDs are commonly used to specify structural relationships of objects in data-centric XML documents, we preserve all the structural information of XSD constructs. We adequately model the complete syntactic relationships of XSD components and fully appreciate how components relate to other ones on all three levels of instance, schema and meta-model.

Generated ontologies are not directly as useful as manually created domain ontologies as they are not conform to the highest quality requirements of more sophisticated domain ontologies regarding the intended semantics of given domains, since XSDs only transport information about (1) the syntactic structure of sets of XML documents, (2) the terminology of individual domains, and (3) implicit semantics of XSD constructs with limited capabilities. By automatically deriving domain ontologies out of these generated ontologies using manually defined SWRL rules, however, we (1) reduce the complexity of generated ontologies and thus underlying XSDs and (2) further supplement OWL axioms with additional domain-specific semantic information not or not satisfyingly covered by XSDs.

By means of a complete case study, we applied the proposed approach and derived DDI-RDF to demonstrate a fast way to disseminate the huge amount of already existing XSDs and XML documents of the commonly accepted and widely adopted XML standards DDI-Codebook and DDI-Lifecycle.

By evaluating the suggested approach, we verified the hypothesis that the effort and the time needed to deliver high quality domain ontologies from scratch by reusing information of already existing XSDs properly describing certain domains is much less than creating domain ontologies completely manually and from the ground up. The resulting domain ontologies are as usable as ontologies that are completely constructed by hand, but with a fraction of necessary effort.

The first step is to transform XSDs into OWL ontologies. To prove the generality of these transformations, we show that any XSD can be converted to OWL by executing generic test cases created out of the XSD meta-model. In addition, we converted XSDs of six widely known, accepted, and used XML standards from the academic (i.a., DDI-Lifecycle) and industrial field. The second step is to define SWRL rules to derive domain ontologies. We specified SWRL rules for three domain ontologies - two from the industrial and one from the academic area. To verify the hypothesis, we determined the effort and expenses for (1) the traditional manual and (2) the suggested semi-automatic approach. DDI-RDF serves as domain ontology, since we were part of the process creating it manually from scratch.

Limitation 1 *XML Schemas must adequately represent particular domains in a syntactically and semantically correct way*

The suggested approach is not suitable when existing XSDs do not properly describe the conceptual model about data of a specific domain in a syn-

tactically and semantically correct manner: (1) syntactically according to the syntactic structure of sets of XML documents and (2) semantically regarding the terminology of the domain and the implicit semantics of XSD constructs.

Research Question 3 *Which types of constraints must be expressible by constraint languages to meet all collaboratively and comprehensively identified requirements to formulate constraints and validate RDF data?*

Our work is supposed to create a stable foundation for subsequent activities in the working groups on RDF validation. We propose to relate existing solutions to case studies and use cases by means of requirements, extracted from the latter and fulfilled by the former. We therefore collected the findings of the RDF Validation Workshop and the working groups and initiated a community-driven database of requirements to formulate constraints and validate RDF data against constraints. Additionally, we added requirements from other sources, particularly in the form of constraint types that are supported by existing approaches, e.g., those expressible in OWL 2. 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.

Laying the ground on case studies collected from many data practitioners and relating solutions to case studies and use cases by means of requirements, makes sure that (1) commonly approved requirements cover real world needs of data professionals having RDF validation related problems and (2) the further development of constraint languages is based on universally accepted requirements.

By today, we have published 81 types of constraints that are required by various 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 a specific requirement in the database (see Chapter 5).

Contribution 3 *Publication of 81 types of constraints that must be expressible by constraint languages to meet all jointly and extensively identified requirements to formulate constraints and validate RDF data against constraints*

We used this collection of constraint types to gain a better understanding of the expressiveness of existing and currently developed solutions, i.e., to evaluate to which extent each requirement is fulfilled by the most common constraint languages, identify gaps that still need to be filled, recommend possible solutions for their elimination, and give directions for the further development of constraint languages.

Research Question 4 *How to ensure for any constraint type that (1) RDF data is consistently validated against semantically equivalent constraints of the same constraint type across RDF-based constraint languages and (2) semantically equivalent constraints of the same constraint type can be transformed from one RDF-based constraint language to another?*

There is not a one-size-fits-all solution: none of the solutions, we consider as high-level constraint languages, is able to meet all requirements raised by data practitioners, i.e., enables to express each of the 81 identified constraint types.

High-level constraint languages like ShEx, ReSh, DSP, OWL, and SHACL either lack an implementation to actually validate RDF data against constraints expressed in these languages or are based on different implementations, which leaves the question how RDF-based constraint languages can be executed on RDF data in a consistent way.

The overall idea is that we see constraint languages as domain-specific languages that are translated into SPARQL and executed on RDF data. We use *SPIN*, a SPARQL-based way to formulate and check constraints, as basic validation framework and present a general approach how the validation of RDF data against constraints of any constraint type can be consistently implemented regardless of the language used to express them (see Chapter 6).

Contribution 4-1 *Consistent validation across RDF-based constraint languages*

We propose to use SPARQL as low-level implementation language: constraint types are transformed into SPARQL queries executable to validate RDF data against constraints instantiated from these constraint types. We claim and provide evidence from literature that constraints of each constraint type expressed in any RDF-based language can be checked with plain SPARQL as execution language.

The translation of a constraint language into SPARQL queries is done once, for instance, by the language designer, and provided in form of a SPIN mapping. The mapping from a constraint language to SPIN is performed by creating SPIN construct templates representing SPARQL CONSTRUCT queries in RDF - one for each constraint type that is supported by the language. SPIN construct templates generate constraint violation triples indicating the subjects, the properties, and the constraints causing the violations and the reasons why violations have been raised. In addition, we may give some guidance how to become valid data and to indicate how severe the violation of a constraint is, we may classify constraints according to different levels of severity like *informational*, *warning*, and *error*. Representing constraint violations and therefore validation results in RDF enables to process them further.

We have developed a validation environment which is online available at http://purl.org/net/rdfval-demo and which can be used to validate RDF data on constraints of any type expressed in arbitrary RDF-based constraint lan-

guages. The SPIN engine checks for each resource if it satisfies all constraints, which are associated with the classes the resource is assigned to, and generates a result RDF graph containing information about all constraint violations.

To demonstrate the general applicability of the approach, we completely implemented the validation of RDF data against all OWL 2 and DSP language constructs by mapping each of them to SPIN. Furthermore, we provide implementations for all constraint types which are expressible in OWL 2 and DSP as well as for major constraint types representable by ReSh and ShEx.

We generalize from our experience implementing high-level constraint languages and gained from the analysis of the identified constraint types and introduce a general framework, an abstraction layer that enables to formulate constraints of any constraint type in a way that mappings from high-level constraint languages to the 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 with a small lightweight vocabulary and constraining elements (see Chapter 7).

Contribution 4-2 *Minimal representation of constraints of any type*

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

Even though SPIN provides a convenient way to represent constraint violations and to validate RDF data, the implementation of a high-level constraint language still requires a tedious mapping to SPIN with a certain degree of freedom how constraints of a certain type are checked and how violations of constraints of a particular type are represented.

Our framework therefore provides a common ground that is solely based on the abstract definitions of constraint types. By providing just one SPIN mapping for each constraint type, it is ensured that the details of the SPIN implementation are consistent irrespective of the language and that the validation against semantically equivalent constraints of the same constraint type leads always to exactly the same results independently of the language used to express them. This means that whenever semantically equivalent constraints in different languages are checked on RDF data they point out the same set of violations.

Contribution 4-3 *For any constraint type, RDF data is consistently validated against semantically equivalent constraints of the same constraint type across RDF-based constraint languages*

As there is no standard way to define constraints, semantically equivalent constraints of the same type are expressible by a variety of constraint languages - each of them different in syntax and semantics. Mappings from constraint languages to the abstraction layer and back enable transformations of semantically equivalent constraints of the same type from one constraint language to another via the intermediate representation, hence increase the interoperability of constraint languages. With an intermediate generic representation, we only need to define $2n$ mappings for each constraint type and not $n \cdot (n-1)$ mappings, i.e., for each constraint type and each possible combination of n constraint languages.

Contribution 4-4 *For any constraint type, semantically equivalent constraints of the same constraint type can be transformed from one RDF-based constraint language to another*

By providing just one SPIN mapping for each constraint type independently of concrete languages, the implementation of constraint types is simplified for existing and newly developed constraint languages. All that is needed to reuse consistent implementations of constraint types is to define bidirectional mappings.

With this implementation, there are two obvious limitations of our approach. The first limitation is that constraint language constructs and constraints must be representable in RDF in order to be able to provide implementations for their constraint types and to validate RDF data according to them within our validation environment. Nevertheless, for each of the 81 published constraint types, instantiated constraints can be represented in RDF.

Limitation 2 *Constraints of supported constraint types and constraint language constructs must be representable in RDF*

To consistently execute RDF-based constraint languages, it is required that they can be expressed in SPARQL, i.e., the actual checking of constraints of each constraint type the language supports must be expressible in form of a SPARQL query. For SHACL, OWL 2, ShEx, DSP, and ReSh, this is the case, but in the future, we would like to support non-RDF based languages as well. It is not necessarily a problem that languages have to be expressible in SPARQL, as the data and the data models are represented in RDF, so this is consistent. As case studies, use cases, and requirements are still collected, it is likely that the list of the up to now 81 constraint types will be expanded. But even if there are constraint types that cannot be translated into SPARQL, the subset of supported constraint types is certainly large enough to justify the limitation to SPARQL-expressible constraint types.

Limitation 3 *Constraint languages and supported constraint types must be expressible in SPARQL*

Research Question 5 *What is the role reasoning plays in practical data validation and for which constraint types reasoning may be performed prior to validation to enhance data quality?*

The set of constraint types forms the basis to investigate the role that reasoning and different semantics play in practical data validation and how to overcome the major shortcomings when validating RDF data by performing reasoning prior to validation (see Chapter 8). Reasoning is beneficial for RDF validation as (1) it may resolve constraint violations, (2) it may cause valuable violations, and (3) it solves the redundancy problem.

We investigated the effect of reasoning to the validation process for each constraint type, i.e., we examined for each constraint type if reasoning may be performed prior to validation to enhance data quality either by resolving violations or by raising valuable violations and solving them. For more than 2/5 of the constraint types, reasoning may be performed to improve data quality. For less than 3/5 of the constraint types, however, reasoning cannot be done or does not improve data quality in any obvious sense.

For reasoning constraint types, different types of reasoning - OWL 2 QL and OWL 2 DL reasoning - may be performed which depends on the language used to formulate the constraint type. We consider these OWL 2 sublanguages, as they differ with regard to expressivity and efficiency in performance.

We examined the effects of reasoning on the performance, i.e., the computational complexity of constraint types. Therefore, we investigated for each constraint type how efficient in terms of runtime validation is performed with and without reasoning. By mapping to Description Logics we get an idea of the performance for each constraint type in worst case, since the combination of Description Logics constructs needed to express a constraint type determines its computational complexity.

Validation and reasoning assume different semantics which may lead to different validation results when applied to particular constraint types. Thus, we investigated for each constraint type (1) if it depends on the CWA and (2) if it depends on the UNA. For the majority of the constraint types, it makes a difference in terms of validation results. For these constraint types, we have to be careful in case we want to use them for reasoning and for validation.

Contribution 5 *We delineate the role reasoning plays in practical data validation and investigated for each constraint type (1) if reasoning may be performed prior to validation to enhance data quality, (2) how efficient in terms of runtime validation is performed with and without reasoning, and (3) if validation results depend on different underlying semantics.*

By revealing which languages do not cover certain constraint types for which reasoning can be done to enhance data quality, we emphasize the significant role SPIN and OWL 2 DL play for the future development of constraint languages.

Evaluation

A concrete constraint is instantiated from a constraint type and defined for a specific vocabulary. We collected 115 constraints on three vocabularies commonly used in the SBE sciences (DDI-RDF, QB, SKOS), either from the vocabularies themselves or from several domain experts, and set up a large-scale evaluation to gain a better understanding about the role of certain constraint types for determining the quality of RDF data and thus to evaluate the usability of constraint types for assessing RDF data quality (see Chapter 9).

Contribution 6 *Evaluation of the Usability of Constraint Types for Assessing RDF Data Quality*

We classified these constraints based on which types of constraint languages enable to express their constraint types, i.e., whether their type is representable by RDFS/OWL, common high-level constraint languages, or only by plain SPARQL. Furthermore, based on the classification system of log messages in software development, we let the domain experts classify the constraints with regard to the perceived severity of their violation.

We conducted a large-scale experiment and evaluated the data quality of 15,694 data sets (4.26 billion triples) of SBE research data, obtained from 33 SPARQL endpoints, by validating the data against these 115 constraints. As the evaluation is based on only three vocabularies, we cannot make valid general statements for all vocabularies, but we can formulate several findings and make recommendations to direct the further development of constraint languages. If the findings hold generally for all vocabularies is still to be determined, nevertheless they provide valuable insights for future developments in this field.

Limitation 4 *The generality of the findings of the large-scale evaluation has to be proved for all vocabularies*

10.2 Future Work

Vocabularies for Representing Research Data and its Metadata

Within two public review phases in 2014 and 2016 for two months in each case, we received valuable feedback on the newly developed vocabularies from members of the Linked Data community, the SBE sciences, the statistical domain, as well as DDI experts and implementers. In the middle of the year 2016, we plan to publish the vocabularies as official DDI Alliance specifications. As there is already a W3C recommendation for QB, discussions are ongoing to work on a respective W3C recommendation for DDI-RDF.

The interplay of DDI-RDF, QB, and PROV-O needs further exploration regarding the relationship of aggregated data, aggregation methods, and the underlying unit-record data. The goal is to drill down to related unit-record

data based on a search resulting in aggregated data. A researcher could then analyze the unit-record data to answer further research questions.

Experiences with DDI-RDF have led to a broader set of requirements which will be met by DDI-Lifecycle MD, the model-driven further development of the DDI standard itself (see Section 3.2). As members to the DDI Moving Forward Project, we will contribute formalizing the conceptual model and implementing diverse model serializations, first and foremost RDFS and OWL. DDI-Lifecycle MD will incorporate the results of ongoing work which focuses on the early phases of survey design and data collection as well as on other data sources like register data. DDI has its strengths in the domain of SBE research data, but will be opened to data of other disciplines as well.

As PHDD and *CSV on the Web (CSVW)* [302] have an overlap in the main description of tabular data in CSV format, efforts should be made aligning these specifications. Nevertheless, the scope of PHDD is broader, since compared to CSVW, PHDD can also be used to describe tabular data with fixed record length and with multiple records per case. Both, PHDD and CSVW will be evaluated for the work on data description in DDI-Lifecycle MD.

RDFication of XML Enabling to use RDF Validation Technologies

We will complete our XSLT framework generating OWL ontologies stylesheet-driven out of XSDs by developing an XSLT which translates XML documents directly without XSDs. The first step is to create suitable XSDs out of XML documents. These XSDs will then be converted to OWL in a second step.

We generalize from XSD meta-model based uni-directional transformations from XSD models into OWL models to bidirectional transformations between models of any meta-model such as OWL 2, XSD, relational database schemas, Java, and UML 2 (see Section 2.3). Meta-model based transformations of arbitrary models to OWL models enable to convert any data to RDF and to validate any data according to constraints extractable from models of arbitrary meta-models using common RDF validation tools.

QVT can be used to define transformation models describing transformations between source and target models of any meta-model. Instead of transforming models directly on the model level, model transformations are specified on the meta-model level using meta-model constructs and semantics. By defining transformations on a higher meta-level, model transformations do not depend on the converted models. They only depend on the involved meta-models which enables an elegant description of the transformation.

EMF provides a basic framework for the model-driven engineering of software systems which supports MOF-based meta-modeling. As part of EMF, ecore has emerged as the de facto standard for the definition of meta-models and therefore serves as meta-meta-model to specify arbitrary meta-models. Prerequisite for performing meta-model based model transformations is to represent meta-models and models as direct and indirect instances of the ecore meta-meta-model. There are multiple MOF-based meta-models which

are already represented conforming to ecore, such as OWL 2, XSD, and UML 2. As model transformations are defined by means of the abstract syntax instead of concrete syntaxes of a language, they become independent of any particular representation and transformations do not have to be defined for each concrete syntax.

We started focusing on unidirectional transformations of XSD models into OWL models based on the XSD meta-model. By means of the generalized meta-model based model transformations, we are able to define transformations between models of any meta-model in both directions. After transformations, it is likely that source and target models change independently from each other on both the schema and the instance level. Thus, it is necessary to synchronize source and target models and their instances continuously. We achieve round tripping by defining mappings between source and target meta-model constructs. After automatically performed round tripping, we do not have to adapt source and target models according to changes of the respective other side manually on the schema and the instance level.

DDI-RDF, DDI-Codebook, and DDI-Lifecycle may serve as case studies to (1) convert XSD to OWL models, (2) translate XML into RDF corresponding to these models, and (3) validate XML on constraints extracted from OWL models. DDI-Lifecycle MD may be used as case study to (1) translate UML models into models of multiple meta-models on the schema and the instance level and (2) validate UML model instances according to UML model constraints when representing them as OWL axioms.

RDF Validation Requirements and Constraint Types

To underpin its importance as a tool for the advancement of constraint formulation and RDF data validation, we will maintain and continuously extend the RDF validation database within the context of the DCMI and the W3C working groups. We will also continue to identify and publish jointly approved requirements and corresponding constraint types to ensure that the further development of constraint languages is based on commonly accepted requirements as well as case studies and use cases collaboratively collected by various data practitioners. Within the DCMI working group, we pursue the establishment of application profiles that allow to link constraints directly to published data sets and ontologies.

Validation Framework for RDF-based Constraint Languages

Based on our experiences providing full implementations for all OWL 2 and DSP constructs and for all constraint types expressible in OWL 2 and DSP, our next steps include the application of the suggested approach to further RDF-based constraint languages, first and foremost ShEx and ReSh, for which we already implemented major constraint types.

To be able to offer implementations for constraint languages within our validation environment, it is mandatory that they can be expressed in SPARQL, i.e., the actual checking of constraints of each constraint type the language supports must be expressible in form of a SPARQL query, which excludes many existing solutions. We will further investigate ways to provide mappings to SPIN, not only for languages expressible in SPARQL but also for non-RDF based constraint languages. Another interesting topic is an automatic testing of SPIN mappings, for which test data together with expected outcomes could be provided in a certain form.

It is part of future work to (1) finalize the consistent implementations for the generic representations of all constraint types, (2) integrate these SPIN mappings in our validation environment which allows users to generically express constraints of each type, (3) fully map constraint languages to the abstraction layer and back enabling transformations of semantically equivalent constraints across these languages, and of course (4) keep the framework and the constraining elements as its core building blocks in sync with the ongoing work in the working groups.

In the majority of cases, domain experts are not capable to express constraints they have in mind using common constraint languages. Because of this, computer scientists or even language designers have to help them defining suitable constraints in languages satisfying their individual needs. In contrast, domain experts are mostly able to represent valid data in RDF, for example, in form of the intuitive and concise concrete RDF Turtle syntax. Therefore, we will investigate ways to follow a reverse engineering approach by automatically generating constraints in the generic intermediate representation out of valid RDF data.

In Section 7.4, we sketched how the framework can be combined with the upcoming W3C recommendation SHACL. As SHACL is still under constant development, the subsequent points are obviously part of future work:

- Derive SHACL extensions with SPARQL bodies for each constraint type not supported by SHACL out of the generic intermediate representation of constraint types and their SPIN mappings including the SPARQL queries to actually check instantiated constraints.
- Enhance the interoperability between SHACL and other high-level constraint languages by defining mappings from SHACL to the abstraction layer and back for each constraint type SHACL supports. This enables transformations of semantically equivalent constraints of the same type between SHACL and other languages.
- Continuously synchronize the implementations of constraint types (1) we generically offer within our framework and (2) SHACL provides. This way, the consistency of the implementations of constraint types among SHACL and other languages is maintained and therefore it is ensured that the validation of RDF data against semantically equivalent constraints of the

same type in different languages leads always to exactly the same results - independently of the language used to express these constraints.

The Role of Reasoning for RDF Validation

We will extend our validation environment to provide a list of languages for which the constraint type specific expressivity is sufficient depending on users' individual needs. The validation environment may also recommend one of these languages covering the most of the required constraint types with the lowest for the user acceptable complexity class determining how efficient in terms of runtime validation is performed. As reasoning may cause high complexity, the validator may show which constraint types from the users' selections cause the higher complexity class and may provide solutions how to get to the next lower class. It would be charming to have an estimation which group of constraint types demands which complexity class. This is not an easy question, however, since complexity results are language specific and operational semantics is involved as well. Therefore, it is hard to maintain a general complexity result for a constraint type independent of the language chosen. Yet, providing an estimation for particular cases can still be straightforward.

Evaluating the Usability of Constraint Types for Assessing RDF Data Quality

To evaluate the usability of constraint types for determining the quality of RDF data, we evaluated the quality of 15,694 data sets (4.26 billion triples) of research data by validating against constraints on three vocabularies. Despite the large volume of the evaluated data sets in general, we have to keep in mind that the evaluation is based on only three vocabularies. Even though we can formulate several findings and make recommendations to direct the future development of constraint languages, we cannot make valid general statements for all vocabularies. As these findings cannot be proved yet, they still have to be verified or falsified by evaluating the quality of RDF data conforming to further well-established and newly developed vocabularies.

10.3 Conclusion

For constraint formulation and RDF data validation, several languages exist or are currently developed. Even with an upcoming W3C recommendation for a new constraint language, it can be expected that several languages will be used in practice – consider the situation in the XML world, where a standardized schema language was available from the beginning and yet additional ways to specify and check constraints have been created.

As the application of constraint languages per se improves data quality, it must be proceeded working intensively on their enhancement. We are actively involved in the further development and implementation of constraint

languages and will incorporate the findings of this thesis into the DCMI and W3C working groups on RDF validation to set priorities on features where we expect the highest impact on the data quality of real-life data in the social, behavioral, and economic sciences and the cultural heritage sector.

The intention of this thesis is to provide a scientifically sound, practice-oriented, and sustainable basis for continued research in the field of RDF validation and the development of constraint languages.

Constraints expressed in common constraint languages are extractable out of RDF vocabularies and XML Schemas. This is the reason why (1) we have developed three missing vocabularies to represent all types of research data and its metadata and to validate RDF data according to constraints extractable from these vocabularies, and (2) we propose a general approach directly validating XML against semantically rich OWL axioms when using them in terms of constraints, extracted from XML Schemas adequately representing particular domains.

We have published a set of constraint types, required by various stakeholders for data applications, which forms the basis of this thesis. Each constraint type corresponds to one of the requirements derived from jointly collected case studies and use cases. We use this collection (1) to gain a better understanding of the expressiveness of existing and currently evolved solutions, (2) to identify gaps that still need to be filled, (3) to investigate which languages cover certain constraint types for which reasoning may be performed prior to validation to enhance data quality, (4) to evaluate the usability of constraint types for assessing RDF data quality, and (5) to give directions for the future development of constraint languages.

We introduce a validation framework as (1) none of the languages, we consider as high-level constraint languages, is able to meet all requirements raised by various data practitioners, i.e., enables to express each constraint type and (2) these languages either lack an implementation to actually validate RDF data on constraints expressed in these languages or are based on different implementations.

The framework 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 with a small lightweight vocabulary. We demonstrate that using another layer on top of SPARQL ensures for each constraint type that (1) the validation of semantically equivalent constraint of the same type points out the same set of violations regardless of the language used to express them and (2) semantically equivalent constraints of the same type can be transformed from one language to another.

As constraints of different types are representable by different languages, we suggest to base the selection of a suitable constraint language on the requirements to be satisfied and to use the framework to express constraint types not supported by chosen languages.

References

[1] Abrial, J.-R. & Schuman, S. A. (1980). A Specification Language. In R. McNaughten & R. C. McKeag (Eds.), *On the Construction of Programs*. Cambridge University Press. http://se.ethz.ch/~meyer/publications/languages/Z_original.pdf.

[2] Alexander, K., Cyganiak, R., Hausenblas, M., & Zhao, J. (2011). *Describing Linked Datasets with the VoID Vocabulary*. W3C Interest Group Note, W3C. http://www.w3.org/TR/2011/NOTE-void-20110303/.

[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] 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.

[5] Angles, R. & Gutierrez, C. (2008). The Expressive Power of SPARQL. In A. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. Finin, & K. Thirunarayan (Eds.), *The Semantic Web - ISWC 2008*, volume 5318 of *Lecture Notes in Computer Science* (pp. 114–129). Springer Berlin Heidelberg. http://link.springer.com/chapter/10.1007%2F978-3-540-88564-1_8.

[6] Anicic, N., Ivezic, N., & Marjanovic, Z. (2007). Mapping XML Schema to OWL. In *Enterprise Interoperability* (pp. 243–252). Springer.

[7] 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.

[8] Aranda, C. B., Corby, O., Das, S., Feigenbaum, L., Gearon, P., Glimm, B., Harris, S., Hawke, S., Herman, I., Humfrey, N., Michaelis, N., Ogbuji, C., Perry, M., Passant, A., Polleres, A., Prud'hommeaux, E., Seaborne, A.,

& Williams, G. T. (2013). *SPARQL 1.1 Overview*. W3C Recommendation, W3C. http://www.w3.org/TR/2013/REC-sparql11-overview-20130321/.

[9] Archer, P. & Shukair, G. (2013). *Asset Description Metadata Schema (ADMS)*. W3C Working Group Note, W3C. http://www.w3.org/TR/2013/NOTE-vocab-adms-20130801/.

[10] Arora, S. & Barak, B. (2009). *Computational Complexity: A Modern Approach*. New York, NY, USA: Cambridge University Press, 1st edition.

[11] Artale, A., Calvanese, D., Kontchakov, R., & Zakharyaschev, M. (2009). The DL-Lite Family and Relations. *Journal of Artificial Intelligence Research*, 36(1), 1–69.

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

[13] Baader, F. & Nutt, W. (2003). The Description Logic Handbook. In F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, & P. F. Patel-Schneider (Eds.), *Basic Description Logics* (pp. 43–95). New York, NY, USA: Cambridge University Press.

[14] Baker, C., Baran, J., Bolleman, J., Callahan, A., Chepelev, L., Cohen, K., Courtot, M., Duck, G., Furlong, L., McCarthy, L., McCusker, J., Cruz-Toledo, J. M., Hoehndorf, R., Jupp, S., Kim, J.-D., Klassen, D., Luetteke, T., Malone, J., Mungall, C., Osumi-Sutherland, D., Ohta, T., Queralt, N., Reed, S., Riazanov, A., Samwald, M., Stevens, R., Wilkinson, M., Verspoor, K., & Villanueva-Rosales, N. (2010). *Semanticscience Integrated Ontology (SIO)*. Specification, Semantic Science. http://semanticscience.org/ontology/sio.owl.

[15] Balzert, H. (2009). *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*. Lehrbücher der Informatik. Springer Spektrum, 3 edition. http://www.springer.com/us/book/9783827417053.

[16] Balzert, H. (2011). *Lehrbuch Der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb*. Lehrbücher der Informatik. Springer Spektrum, 3 edition. http://www.springer.com/us/book/9783827417060.

[17] Battle, S. (2006). Gloze: XML to RDF and Back Again. In *Proceedings of the 1st Jena User Conference* Bristol, UK.

[18] Berners-Lee, T. (1996). WWW: Past, Present, and Future. *IEEE Computer*, 29(10), 69–77. http://doi.ieeecomputersociety.org/10.1109/2.539724.

[19] Berners-Lee, T. (1997). World-Wide Computer. *Communications of the ACM*, 40(2), 57–58. http://cacm.acm.org/magazines/1997/2/8461-world-wide-computer/abstract.

[20] Berners-Lee, T., Cailliau, R., Groff, J. F., & Pollermann, B. (1992). World-Wide Web: The Information Universe. *Electronic Networking: Research, Applications and Policy*, 1(2), 74–82.

[21] Berners-Lee, T., Cailliau, R., Luotonen, A., Nielsen, H. F., & Secret, A. (1994). The World-Wide Web. *Communications of the ACM*, 37(8), 76–82. http://doi.acm.org/10.1145/179606.179671.

[22] Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web. *Scientific American*, 284(5), 29–37. http://www.scientificamerican.com/article/the-semantic-web/.

[23] Binstock, C., Peterson, D., Smith, M., Wooding, M., Dix, C., & Galtenberg, C. (2002). *The XML Schema Complete Reference*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. http://www.xmlschemareference.com/XMLSchemaCompleteReference.html.

[24] Biron, P. V. & Malhotra, A. (2004). *XML Schema Part 2: Datatypes Second Edition*. W3C Recommendation, W3C. http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/.

[25] Blank, G. & Rasmussen, K. B. (2004). The Data Documentation Initiative: The Value and Significance of a Worldwide Standard. *Social Science Computer Review*, 22(3), 307–318.

[26] 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*. http://www.ddialliance.org/system/files/DevelopingaModel-DrivenDDISpecification2013_05_15.pdf.

[27] Bock, C., Fokoue, A., Haase, P., Hoekstra, R., Horrocks, I., Ruttenberg, A., Sattler, U., & Smith, M. (2012). *OWL 2 Web Ontology Language - Structural Specification and Functional-Style Syntax (Second Edition)*. W3C Recommendation, W3C. http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/.

[28] Bohr, J. (2007). *Final Report MISSY User Survey*. ZUMA Methodenbericht 01, Gesis - Leibniz Institute for the Social Sciences, Mannheim, Germany. http://www.gesis.org/fileadmin/upload/forschung/publikationen/gesis_reihen/gesis_methodenberichte/2007/07_01_Bohr.pdf.

[29] Bohr, J., Hopt, O., Lengerer, A., Schroedter, J., & Wira-Alam, A. (2010). *Microdata Information System MISSY: Metadata for the Microcensus Scientific Use Files (Final Report MISSY II)*. Gesis Technical Report 07, Gesis - Leibniz Institute for the Social Sciences, Mannheim, Germany. http://www.gesis.org/fileadmin/upload/forschung/publikationen/gesis_reihen/gesis_methodenberichte/2010/TechnicalReport_10-7.pdf.

[30] Bohr, J., Janßen, A., Lengerer, A., Lüttinger, P., Schroedter, J., & Wolf, C. (2007). *Improvement of Access to Microdata for Research. Pilot Project for the Construction of a Service Center for Microdata of GESIS at ZUMA*. ZUMA Methodenbericht 05, Gesis - Leibniz Institute for the Social Sciences, Mannheim, Germany. http://www.gesis.org/fileadmin/upload/forschung/publikationen/gesis_reihen/gesis_methodenberichte/2007/07_05_bohr_www.pdf.

[31] Bohr, J., Janßen, A., & Wackerow, J. (2006). Problems of Comparability in the German Microcensus over Time and the New DDI Version 3.0. *IAS-*

SIST Quarterly, 30(2), 13–19. http://www.iassistdata.org/sites/default/files/iq/iqvol301bohr.pdf.

[32] Bohring, H. & Auer, S. (2005). Mapping XML to OWL Ontologies. In *Leipziger Informatik-Tage, Volume 72 of LNI* (pp. 147–156). Leipzig, Germany: GI.

[33] Boneva, I., Gayo, J. E. L., Hym, S., Prud'hommeau, E. G., Solbrig, H. R., & Staworko, S. (2014). Validating RDF with Shape Expressions. *Computing Research Repository (CoRR)*, abs/1404.1270.

[34] Boneva, I., Gayo, J. E. L., Prud'hommeaux, E. G., & Staworko, S. (2015). Shape Expressions Schemas. *Computing Research Repository (CoRR)*, abs/1510.05555. http://arxiv.org/abs/1510.05555.

[35] Boneva, I. & Prud'hommeaux, E. (2015). *Core SHACL Semantics*. W3C Editor's Draft, W3C. http://w3c.github.io/data-shapes/semantics/.

[36] Börger, E. & Rosenzweig, D. (1995). A Mathematical Definition of Full Prolog. *Science of Computer Programming*, 24(3), 249–286. http://dx.doi.org/10.1016/0167-6423(95)00006-E.

[37] Bos, B. (1997). *The XML Data Model*. Technical report, W3C. http://www.w3.org/XML/Datamodel.html.

[38] **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.

[39] **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://dx.doi.org/10.1145/2814864.2814867.

[40] **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/.

[41] **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.

[42] **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.

[43] **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.

[44] **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.

[45] **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.

[46] **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/.

[47] **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.

[48] **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/.

[49] **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.

[50] **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.

[51] **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.

[52] **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.

[53] **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.

[54] **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.

[55] **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.

[56] **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.

[57] Boucké, N., Weyns, D., Hilliard, R., Holvoet, T., & Helleboogh, A. (2008). Characterizing Relations between Architectural Views. In R. Morrison, D. Balasubramaniam, & K. Falkner (Eds.), *Software Architecture*, volume 5292 of *Lecture Notes in Computer Science* (pp. 66–81). Springer Berlin Heidelberg.

[58] Bourret, R., Cowan, J., Macherius, I., & St. Laurent, S. (1999). *Document Definition Markup Language (DDML) Specification.* W3C Note, W3C. https://www.w3.org/TR/1999/NOTE-ddml-19990119.

[59] Bray, T., Frankston, C., & Malhotra, A. (1998). *Document Content Description for XML.* W3C Note, W3C. https://www.w3.org/TR/1998/NOTE-dcd-19980731.

[60] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., & Yergeau, F. (2008). *Extensible Markup Language (XML) 1.0 (Fifth Edition).* W3C Recommendation, W3C. http://www.w3.org/TR/2008/REC-xml-20081126/.

[61] Brickley, D. & Guha, R. V. (2014). *RDF Schema 1.1.* W3C Recommendation, W3C. http://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

[62] Brickley, D. & Miller, L. (2014). *FOAF Vocabulary Specification 0.99.* Specification. http://xmlns.com/foaf/spec/20140114.html.

[63] Brockmans, S. (2007). *Metamodel-based Knowledge Representation.* PhD thesis, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany.

[64] Brockmans, S., Colomb, R., Haase, P., Kendall, E., Wallace, E., Welty, C., & Xie, G. (2006a). A Model Driven Approach for Building OWL DL and OWL Full Ontologies. In I. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, & L. Aroyo (Eds.), *The Semantic Web - ISWC 2006*, volume 4273 of *Lecture Notes in Computer Science* (pp. 187–200). Springer Berlin Heidelberg.

[65] Brockmans, S. & Haase, P. (2006a). *A Metamodel and UML Profile for Networked Ontologies - A Complete Reference*. Technical report, Universität Karlsruhe (TH).

[66] Brockmans, S. & Haase, P. (2006b). *A Metamodel and UML Profile for Rule-extended OWL DL Ontologies - A Complete Reference*. Technical report, Universität Karlsruhe (TH).

[67] Brockmans, S., Haase, P., Hitzler, P., & Studer, R. (2006b). A Metamodel and UML Profile for Rule-Extended OWL DL Ontologies. In Y. Sure & J. Domingue (Eds.), *The Semantic Web: Research and Applications*, volume 4011 of *Lecture Notes in Computer Science* (pp. 303–316). Springer Berlin Heidelberg.

[68] Brockmans, S., Haase, P., & Stuckenschmidt, H. (2006c). Formalism-Independent Specification of Ontology Mappings - A Metamodeling Approach. In R. Meersman & Z. Tari (Eds.), *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, volume 4275 of *Lecture Notes in Computer Science* (pp. 901–908). Springer Berlin Heidelberg.

[69] Brockmans, S., Haase, P., & Studer, R. (2006d). A MOF-based Metamodel and UML Syntax for Networked Ontologies. In *5th International Semantic Web Conference (ISWC), 2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE)*.

[70] Brockmans, S., Volz, R., Eberhart, A., & Löffler, P. (2004). Visual Modeling of OWL DL Ontologies Using UML. In S. A. McIlraith, D. Plexousakis, & F. van Harmelen (Eds.), *The Semantic Web - ISWC 2004*, volume 3298 of *Lecture Notes in Computer Science* (pp. 198–213). Springer Berlin Heidelberg.

[71] Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., & Grose, T. J. (2003). *Eclipse Modeling Framework: A Developer's Guide*. The Eclipse Series. Addison-Wesley Professional.

[72] Bussler, C. (2003). *B2B Integration: Concepts and Architecture*. Springer-Verlag Berlin Heidelberg.

[73] Bézivin, J. (2006). Model Driven Engineering: An Emerging Technical Space. In R. Lämmel, J. Saraiva, & J. Visser (Eds.), *Generative and Transformational Techniques in Software Engineering*, volume 4143 of *Lecture Notes in Computer Science* (pp. 36–64). Springer Berlin Heidelberg.

[74] Cadoli, M. & Lenzerini, M. (1994). The Complexity of Propositional Closed World Reasoning and Circumscription. *Journal of Computer and System Sciences*, 48(2), 255–310.

[75] Calvanese, D., Giacomo, G., Lembo, D., Lenzerini, M., & Rosati, R. (2007). Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family. *Journal of Automated Reasoning*, 39(3), 385–429.

[76] Campbell, C. E., Eisenberg, A., & Melton, J. (2003). XML Schema. *ACM SIGMOD (Special Interest Group on Management of Data) Record*, 32(2), 96–101. http://doi.acm.org/10.1145/776985.777002.

[77] Ceri, S., Gottlob, G., & Tanca, L. (1989). What You Always Wanted to Know About Datalog (And Never Dared to Ask). *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 1(1), 146–166.

[78] Channin, D. S., Mongkolwat, P., Kleper, V., Sepukar, K., & Rubin, D. L. (2010). The caBIG Annotation and Image Markup Project. *Journal of Digital Imaging: the Official Journal of the Society for Computer Applications in Radiology*, 23(2), 217–225.

[79] Clark, J. (2001a). *The Design of RELAX NG*. Technical report, Thai Open Source Software Center Ltd. http://www.thaiopensource.com/relaxng/design.html.

[80] Clark, J. (2001b). *TREX - Tree Regular Expressions for XML Language Specification*. Specification, Thai Open Source Software Center Ltd. http://thaiopensource.com/trex/spec.html.

[81] Clark, J., Cowan, J., Fitzgerald, M., Kawaguchi, J., Lubell, J., Murata, M., Walsh, N., & Webber, D. (2008). *ISO/IEC 19757-2:2008 - Information technology – Document Schema Definition Language (DSDL) – Part 2: Regular-grammar-based validation – RELAX NG*. ISO Standard, International Organization for Standardization (ISO). http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=52348.

[82] Clark, K. & Sirin, E. (2013). On RDF Validation, Stardog ICV, and Assorted Remarks. In *W3C RDF Validation Workshop. Practical Assurances for Quality RDF Data* Cambridge, MA, USA. http://www.w3.org/2012/12/rdf-val/.

[83] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P., & Stafford, J. (2010). *Documenting Software Architectures: Views and Beyond*. Addison-Wesley Professional, 2nd edition.

[84] Cotton, F. (2015). *XKOS - A SKOS Extension for Representing Statistical Classifications*. DDI Alliance Specification, DDI Alliance. http://rdf-vocabulary.ddialliance.org/xkos.html.

[85] Cotton, F., Cyganiak, R., Grim, R., Gillman, D. W., Jaques, Y., & Thomas, W. (2013). XKOS: An SKOS Extension for Statistical Classifications. In *Proceedings of the 59th World Statistics Congress of the International Statistical Institute* The Hague, The Netherlands. http://2013.isiproceedings.org/Files/CPS203-P32-S.pdf.

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

[87] Cyganiak, R., Field, S., Gregory, A., Halb, W., & Tennison, J. (2010). Semantic Statistics: Bringing Together SDMX and SCOVO. In C. Bizer, T. Heath, T. Berners-Lee, & M. Hausenblas (Eds.), *Proceedings of the International World Wide Web Conference (WWW 2010), Workshop on Linked Data on the Web*, volume 628 of *CEUR Workshop Proceedings*. http://ceur-ws.org/Vol-628/ldow2010_paper03.pdf.

[88] Cyganiak, R. & Reynolds, D. (2014). *The RDF Data Cube Vocabulary*. W3C Recommendation, W3C. http://www.w3.org/TR/2014/REC-vocab-data-cube-20140116/.

[89] Cyganiak, R., Wood, D., & Lanthaler, M. (2014). *RDF 1.1 Concepts and Abstract Syntax*. W3C Recommendation, W3C. http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[90] Cyganiak, R., Zhao, J., Alexander, K., & Hausenblas, M. (2011). *Vocabulary of Interlinked Datasets (VoID)*. Digital Enterprise Research Institute (DERI) Specification, Digital Enterprise Research Institute (DERI). http://vocab.deri.ie/void.

[91] Cyganiak, Richard and Dollin, Chris and Reynolds, Dave (2010). *Expressing Statistical Data in RDF with SDMX-RDF*. Technical report. http://publishing-statistical-data.googlecode.com/svn-history/r50/trunk/specs/src/main/html/index.html.

[92] Davidson, A., Fuchs, M., Hedin, M., Jain, M., Koistinen, J., Lloyd, C., Maloney, M., & Schwarzhof, K. (1999). *Schema for Object-Oriented XML 2.0*. W3C Note, W3C. https://www.w3.org/1999/07/NOTE-SOX-19990730/.

[93] DDI Alliance (2013). *Data Documentation Initiative (DDI) Codebook 2.5.1*. DDI Alliance Specification, DDI Alliance. http://www.ddialliance.org/Specification/DDI-Codebook/2.5/.

[94] DDI Alliance (2014). *Data Documentation Initiative (DDI) Lifecycle 3.2*. DDI Alliance Specification, DDI Alliance. http://www.ddialliance.org/Specification/DDI-Lifecycle/3.2/.

[95] DDI Alliance (2015a). *Data Documentation Initiative (DDI) Specification*. DDI Alliance Specification, DDI Alliance. http://www.ddialliance.org/Specification.

[96] DDI Alliance (2015b). *DDI-Lifecycle MD (Model-Driven)*. DDI Alliance Development Draft, DDI Alliance. http://www.ddialliance.org/ddi-moving-forward-process-summary.

[97] de Bruijn, J. & Heymans, S. (2007). Logical Foundations of (e)RDF(S): Complexity and Reasoning. In K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, & P. Cudré-Mauroux (Eds.), *The Semantic Web*, volume 4825 of *Lecture Notes in Computer Science* (pp. 86–99). Springer Berlin Heidelberg.

[98] Debray, S. K. & Mishra, P. (1988). Denotational and Operational Semantics for Prolog. *Journal of Logic Programming*, 5(1), 61–91. http://dx.doi.org/10.1016/0743-1066(88)90007-6.

[99] Decker, S., Melnik, S., van Harmelen, F., Fensel, D., Klein, M., Broekstra, J., Erdmann, M., & Horrocks, I. (2000). The Semantic Web: the Roles of XML and RDF. *Internet Computing, IEEE*, 4(5), 63–73.

[100] Dell'Erba, M., Fodor, O., Ricci, F., & Werthner, H. (2003). Harmonise: A Solution for Data Interoperability. In J. Monteiro, P. Swatman, & L. Tavares (Eds.), *Towards the Knowledge Society*, volume 105 of *IFIP*

- *The International Federation for Information Processing* (pp. 433–445). Springer US.

[101] Dimou, A., Kontokostas, D., Freudenberg, M., Verborgh, R., Lehmann, J., Mannens, E., Hellmann, S., & de Walle, R. V. (2015). Assessing and Refining Mappings to RDF to Improve Dataset Quality. In *Proceedings of the 14th International Semantic Web Conference (ISWC 2015)*. http://iswc2015.semanticweb.org/sites/iswc2015.semanticweb.org/files/93670111.pdf.

[102] Dodds, L. (2001). *Schemarama*. Technical report, O'Reilly Media, Inc. http://www.xml.com/pub/a/2001/02/07/schemarama.html.

[103] Dublin Core Metadata Initiative (DCMI) (2012a). *DCMI Metadata Terms*. DCMI Specification, Dublin Core Metadata Initiative (DCMI). http://dublincore.org/documents/2012/06/14/dcmi-terms/.

[104] Dublin Core Metadata Initiative (DCMI) (2012b). *Dublin Core Metadata Element Set, Version 1.1*. DCMI Specification, Dublin Core Metadata Initiative (DCMI). http://dublincore.org/documents/2012/06/14/dces/.

[105] Duerst, M. and Suignard, M. (2005). *Internationalized Resource Identifiers (IRIs)*. Proposed Standard, Internet Engineering Task Force (IETF). http://tools.ietf.org/html/rfc3987.

[106] Dumontier, M., Baker, C. J., Baran, J., Callahan, A., Chepelev, L., Cruz-Toledo, J., Del Rio, N., Duck, G., Furlong, L., Keath, N., Klassen, D., McCusker, J., Queralt-Rosinach, N., Samwald, M., Villanueva-Rosales, N., Wilkinson, M., & Hoehndorf, R. (2014). The Semanticscience Integrated Ontology (SIO) for Biomedical Research and Knowledge Discovery. *Journal of Biomedical Semantics*, 5(1).

[107] Ecma International (2013). *The JSON Data Interchange Format*. Ecma Standard, Ecma International. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf.

[108] Fallside, C, D. & Walmsley, P. (2004). *XML Schema Part 0: Primer Second Edition*. W3C Recommendation, W3C. http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/.

[109] Ferdinand, M., Zirpins, C., & Trastour, D. (2004). Lifting XML Schema to OWL. In N. Koch, P. Fraternali, & M. Wirsing (Eds.), *Web Engineering*, volume 3140 of *Lecture Notes in Computer Science* (pp. 354–358). Springer Berlin Heidelberg.

[110] Fischer, P. M., Lausen, G., Schätzle, A., & Schmidt, M. (2015). RDF Constraint Checking. In *Proceedings of the Workshops of the EDBT/ICDT 2015 Joint Conference (EDBT/ICDT)*, volume 1330 (pp. 205–212). Brussels, Belgium. http://ceur-ws.org/Vol-1330/.

[111] Fokoue, A. & Ryman, A. (2013). OSLC Resource Shape: A Linked Data Constraint Language. In *W3C RDF Validation Workshop. Practical Assurances for Quality RDF Data* Cambridge, MA, USA. http://www.w3.org/2012/12/rdf-val/.

[112] Fowler, M. (2010). *Domain-Specific Languages*. Addison-Wesley Signature Series. Pearson Education. https://books.google.de/books?id=ri1muolw_YwC.

[113] Frankel, D. S. (2003). *Model Driven Architecture: Applying MDA to Enterprise Computing*. New York, NY, USA: John Wiley & Sons, Inc.

[114] Frankston, C. & Thompson, H. S. (1998). *XML-Data Reduced*. Technical report. http://www.ltg.ed.ac.uk/~ht/XMLData-Reduced.htm.

[115] Fürber, C. & Hepp, M. (2010). Using SPARQL and SPIN for Data Quality Management on the Semantic Web. In W. Abramowicz & R. Tolksdorf (Eds.), *Business Information Systems*, volume 47 of *Lecture Notes in Business Information Processing* (pp. 35–46). Springer Berlin Heidelberg.

[116] Gallaire, H., Minker, J., & Nicolas, J.-M. (1984). Logic and Databases: A Deductive Approach. *ACM Computing Surveys (CSUR)*, 16(2), 153–185. http://doi.acm.org/10.1145/356924.356929.

[117] Gandon, F. & Schreiber, G. (2014). *RDF 1.1 XML Syntax*. W3C Recommendation, W3C. http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/.

[118] García, R., Perdrix, F., & Gil, R. (2006). Ontological Infrastructure for a Semantic Newspaper. In *15th International World Wide Web Conference (WWW), International Workshop on Semantic Web Annotations for Multimedia (SWAMM)*.

[119] Gayo, J., Prud'hommeaux, E., Boneva, I., Staworko, S., Solbrig, H., & Hym, S. (2015). Towards an RDF Validation Language Based on Regular Expression Derivatives. In *Proceedings of the Workshops of the EDBT/ICDT 2015 Joint Conference (EDBT/ICDT)*, volume 1330 (pp. 197–204). Brussels, Belgium. http://ceur-ws.org/Vol-1330/.

[120] Gayo, J. E. L., Prud'hommeaux, E., Solbrig, H. R., & Rodríguez, J. M. l. (2014). Validating and Describing Linked Data Portals using RDF Shape Expressions. In *Proceedings of the 1st Workshop on Linked Data Quality (LDQ 2014), 10th International Conference on Semantic Systems (SEMANTiCS 2014)*, volume 1215 Leipzig, Germany. http://ceur-ws.org/Vol-1215/paper-06.pdf.

[121] Gayo, J. E. L. & Rodríguez, J. M. l. (2013). Validating Statistical Index Data Represented in RDF using SPARQL queries. In *W3C RDF Validation Workshop. Practical Assurances for Quality RDF Data* Cambridge, MA, USA. http://www.w3.org/2012/12/rdf-val/.

[122] Gil, Y. & Miles, S. (2013). *PROV Model Primer*. W3C Working Group Note, W3C. http://www.w3.org/TR/2013/NOTE-prov-primer-20130430/.

[123] Gillman, D. W., Cotton, F., & Jaques, Y. (2013). *eXtended Knowledge Organization System (XKOS)*. Work Session on Statistical Metadata, United Nations Economic Commission for Europe (UNECE), Geneva, Switzerland. http://www.unece.org/fileadmin/DAM/stats/documents/ece/ces/ge.40/2013/WP10.pdf.

[124] Glimm, B. & Ogbuji, C. (2013). *SPARQL 1.1 Entailment Regimes*. W3C Recommendation, W3C. http://www.w3.org/TR/2013/REC-sparql11-entailment-20130321/.

[125] Godby, Carol Jean and Denenberg, Ray (2015). *Common Ground: Exploring Compatibilities Between the Linked Data Models of the Library of Congress and OCLC*. Technical report, Library of Congress. http://www.oclc.org/research/publications/2015/oclcresearch-loc-linked-data-2015.html.

[126] Gosling, J., Joy, B., Steele, G., Bracha, G., & Buckley, A. (2015). *The Java Language Specification - Java SE 8 Edition*. Oracle Specification, Oracle. https://docs.oracle.com/javase/specs/jls/se8/html/index.html.

[127] Green, A. & Humphrey, C. (2013). Building the DDI. *IASSIST Quarterly*, 37(1), 36–44. http://iassistdata.org/downloads/iqvol371_4_green.pdf.

[128] Gregory, Arofan (2011a). *Open Data and Metadata Standards: Should We Be Satisfied with "Good Enough"?* Technical report, Open Data Foundation. http://odaf.org/papers/OpenDataandMetadataStandards.pdf.

[129] Gregory, Arofan (2011b). *The Data Documentation Initiative (DDI): An Introduction for National Statistical Institutes*. Technical report, Open Data Foundation. http://odaf.org/papers/DDI_Intro_forNSIs.pdf.

[130] Gregory, Arofan and Heus, Pascal (2007). *DDI and SDMX: Complementary, Not Competing, Standards*. Open Data Foundation Paper, Open Data Foundation. http://odaf.org/papers/DDI_and_SDMX.pdf.

[131] Gulbransen, D. (2001). *Special Edition Using XML Schema*. Indianapolis, IN, USA: Que Publishing. http://dl.acm.org/citation.cfm?id=516126.

[132] Haarslev, V. & Müller, R. (2001). RACER System Description. In *Automated Reasoning* (pp. 701–705). Springer.

[133] Hansen, J. B., Beveridge, A., Farmer, R., Gehrmann, L., Gray, A. J. G., Khutan, S., Robertson, T., & Val, J. (2015). Validata: An Online Tool for Testing RDF Data Conformance. In *Proceedings of the International Conference on Semantic Web Applications and Tools for Life Sciences (SWAT4LS)* Camridge, England. http://www.swat4ls.org/wp-content/uploads/2015/10/SWAT4LS_2015_paper_3.pdf.

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

[135] **Hartmann, Thomas** (2016a). *Validation Framework for RDF-based Constraint Languages - PhD Thesis Appendix*. Karlsruhe Institute of Technology (KIT), Karlsruhe. http://dx.doi.org/10.5445/IR/1000054062.

[136] **Hartmann, Thomas** (2016b). *Validation Framework for RDF-based Constraint Languages - PhD Thesis Research Data*. Karlsruhe Institute of Technology (KIT), Karlsruhe. http://dx.doi.org/10.5445/BWDD/11.

[137] **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.

[138] **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.

[139] **Hartmann, Thomas**, Zapilko, B., Wackerow, J., & Eckert, K. (2016a). Directing the Development of Constraint Languages by Checking Constraints on RDF Data. *International Journal of Semantic Computing*, 10(02). http://www.worldscientific.com/worldscinet/ijsc.

[140] **Hartmann, Thomas**, Zapilko, B., Wackerow, J., & Eckert, K. (2016b). 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. https://dx.doi.org/10.1109/ICSC.2016.43.

[141] Hausenblas, M., Ayers, D., Feigenbaum, L., Heath, T., Halb, W., & Raimond, Y. (2012). *The Statistical Core Vocabulary (SCOVO)*. DERI Specification, Digital Enterprise Research Institute (DERI). http://vocab.deri.ie/scovo.

[142] Hausenblas, M., Halb, W., Raimond, Y., Feigenbaum, L., & Ayers, D. (2009). SCOVO: Using Statistics on the Web of Data. In *Proceedings of the 6th European Semantic Web Conference on The Semantic Web: Research and Applications (ESWC 2009)* (pp. 708–722). Berlin, Heidelberg: Springer-Verlag.

[143] Hayes, P. J. & Patel-Schneider, P. F. (2014). *RDF 1.1 Semantics*. W3C Recommendation, W3C. http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/.

[144] Heath, T. & Bizer, C. (2011). *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool, 1st edition.

[145] Hendler, J., Berners-Lee, T., & Miller, E. (2002). Integrating Applications on the Semantic Web. *Journal of the Institute of Electrical Engineers of Japan*, 122(10), 676–680. https://www.w3.org/2002/07/swint.

[146] Hilliard, R. (1996). Representing Software Systems Architectures. In *Joint Proceedings of the 2nd International Software Architecture Workshop (ISAW-2) and International Workshop on Multiple Perspectives in Software Development (Viewpoints '96) on SIGSOFT '96 Workshops*, ISAW '96 (pp. 31–34). New York, NY, USA: ACM.

[147] Hilliard, R. (1999a). Views and Viewpoints in Software Systems Architecture. In *Proceedings of the 1st Working IFIP Conference on Software Architecture* San Antonio, TX, USA. http://web.mit.edu/richh/www/writings/hilliard99-ifip.pdf.

[148] Hilliard, R. (1999b). Views as Modules. In *Proceedings of the 1th Workshop on Multi-Dimensional Separation of Concerns in Object-Oriented Systems, Object-oriented Programming, Systems, Languages, and Applications (OOPSLA)* Denver, Colorado, USA. http://www.cs.ubc.ca/~murphy/multid-workshop-oopsla99/position-papers/ws08-hilliard.pdf.

[149] Hilliard, R. (2000). Views as Modules. In *Proceedings of the 4th International Software Architecture Workshop (ISAW-4)* Limerick, Ireland. http://www.cs.huji.ac.il/~wyaron/papers/ISAW_page136.pdf.

[150] Hilliard, R. (2001). "Don't Ask, Don't Tell" Inference: Architectural Views and their Interfaces. In *Proceedings of the 2nd International Workshop on Living with Inconsistency* Toronto, Canada . http://web.mit.edu/richh/www/writings/hilliard01c-dont-ask-dont-tell.pdf.

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

[152] Horridge, M. & Patel-Schneider, P. F. (2012). *OWL 2 Web Ontology Language Manchester Syntax (Second Edition)*. W3C Working Group Note, W3C. http://www.w3.org/TR/2012/NOTE-owl2-manchester-syntax-20121211/.

[153] Horrocks, I., Motik, B., & Wang, Z. (2012). The HermiT OWL Reasoner. In *Proceedings of the 1st International Workshop on OWL Reasoner Evaluation (ORE 2012)* Manchester, UK.

[154] Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., & Dean, M. (2004). *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. W3C Member Submission, W3C. http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/.

[155] Hosoya, H. & Pierce, B. C. (2000). XDuce: A Typed XML Processing Language. In *Proceedings of the 3rd International Workshop on the Web and Databases (WebDB 2000)* (pp. 111–116). http://dblp.uni-trier.de/db/conf/webdb/webdb2000.html.

[156] Hosoya, H. & Pierce, B. C. (2003). XDuce: A Statically Typed XML Processing Language. *ACM Transactions on Internet Technology (TOIT)*, 3(2), 117–148. http://doi.acm.org/10.1145/767193.767195.

[157] Huang, S. S., Green, T. J., & Loo, B. T. (2011). Datalog and Emerging Applications: An Interactive Tutorial. In *Proceedings of the 2011 ACM SIGMOD/PODS Conference (SIGMOD 2011)* (pp. 1213–1216). New York, NY, USA: ACM. http://doi.acm.org/10.1145/1989323.1989456.

[158] Hull, E., Jackson, K., & Dick, J. (2011). *Requirements Engineering*. Springer-Verlag London, 3 edition. http://link.springer.com/book/10.1007%2F978-1-84996-405-0.

[159] Isaac, A. & Summers, E. (2009). *SKOS Simple Knowledge Organization System Primer*. W3C Working Group Note, W3C. http://www.w3.org/TR/2009/NOTE-skos-primer-20090818/.

[160] ISO (1986). *ISO 8879:1986 - Information Processing – Text and Office Systems – Standard Generalized Markup Language (SGML)*. ISO Standard, ISO. http://www.iso.org/iso/catalogue_detail.htm?csnumber=16387.

[161] ISO (2013). *ISO 17369:2013 - Statistical Data and Metadata eXchange (SDMX)*. ISO Standard, ISO. http://www.iso.org/iso/catalogue_detail.htm?csnumber=52500.

[162] ISO (2014). *ISO 19115-1:2014 - Geographic information – Metadata – Part 1: Fundamentals*. ISO Standard, ISO. http://www.iso.org/iso/catalogue_detail.htm?csnumber=53798.

[163] ISO/IEC (2002a). *ISO/IEC 13568:2002 - Information Technology - Z Formal Specification Notation - Syntax, Type System and Semantics*. ISO/IEC Standard, ISO/IEC. http://standards.iso.org/ittf/PubliclyAvailableStandards/c021573_ISO_IEC_13568_2002(E).zip.

[164] ISO/IEC (2002b). *ISO/IEC TR 22250-1:2002 - Information Technology – Document Description and Processing Languages – Regular Language Description for XML (RELAX) – Part 1: RELAX Core*. ISO/IEC Standard, ISO/IEC. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=34922.

[165] ISO/IEC (2004a). *ISO/IEC 11179-1:2004(E) - ISO/IEC 11179, Information Technology – Metadata registries (MDR)*. ISO/IEC Standard, ISO/IEC. http://metadata-standards.org/11179/.

[166] ISO/IEC (2004b). *ISO/IEC 19757-1:2004 - Information Technology - Document Schema Definition Languages (DSDL) – Part 1: Overview*. ISO/IEC Standard, ISO/IEC.

[167] ISO/IEC (2006). *ISO/IEC 19757-3:2006 - Information Technology — Document Schema Definition Languages (DSDL) - Part 3: Rule-Based Validation - Schematron*. ISO/IEC Standard, ISO/IEC. http://standards.iso.org/ittf/PubliclyAvailableStandards/c040833_ISO_IEC_19757-3_2006(E).zip.

[168] Jacobs, J. A. & Humphrey, C. (2004). Preserving Research Data. *Communications of the ACM*, 47(9), 27–29. http://doi.acm.org/10.1145/1015864.1015881.

[169] Jain, P., Hitzler, P., Yeh, P. Z., Verma, K., & Sheth, A. P. (2010). Linked Data Is Merely More Data. In *AAAI Spring Symposium: Linked Data Meets Artificial Intelligence*: AAAI.

[170] Janßen, A. & Bohr, J. (2006). Microdata Information System – MISSY. *IASSIST Quarterly*, 30(2), 5–11. http://www.iassistdata.org/sites/default/files/iq/iqvol302bohr.pdf.

[171] Jelliffe, R. (2001). The Current State of the Art of Schema Languages for XML. In *Proceedings of the OASIS Open Standards XML Asia Pacific Conference* Sydney, Australia. http://chinese-school.netfirms.com/articles/RickJelliffe.pdf.

[172] Jelliffe, R. (2002). *Resource Directory (RDDL) for Hook 0.2 - A One-Element Language for Validation of XML Documents based on Partial Order*. Specification, Academia Sinica Computing Centre. http://xml.ascc.net/hook/.

[173] Johnson, D. & Speicher, S. (2013). *Open Services for Lifecycle Collaboration Core Specification Version 2.0*. OSLC Specification, Open Services for Lifecycle Collaboration (OSLC). http://open-services.net/bin/view/Main/OslcCoreSpecification.

[174] Kaukonen, E. & Leino, J. (2015). Implementing the GSIM Statistical Classification Model – the Finnish Way. In *Proceedings of the Workshop on International Collaboration for Standards-Based Modernisation, United Nations Economic Commission for Europe Conference of European Statisticians* Geneva, Switzerland. http://www1.unece.org/stat/platform/display/WICSBM/Geneva%2C+5-7+May+2015.

[175] Kauppinen, T., Baglatzi, A., & Keßler, C. (2013). Linked Science: Interconnecting Scientific Assets. In T. Critchlow & K. Kleese-Van Dam (Eds.), *Data Intensive Science*. USA: CRC Press.

[176] Kay, M. (2007). *XSL Transformations (XSLT) Version 2.0*. W3C recommendation, W3C. http://www.w3.org/TR/2007/REC-xslt20-20070123/.

[177] Kifer, M. (2005). Rules and Ontologies in F-Logic. In N. Eisinger & J. Małuszyński (Eds.), *Reasoning Web*, volume 3564 of *Lecture Notes in Computer Science* (pp. 22–34). Springer Berlin Heidelberg.

[178] Kifer, M., Lausen, G., & Wu, J. (1995). Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the ACM*, 42(4), 741–843.

[179] Klarlund, N., Møller, A., & Schwartzbach, M. I. (2000). DSD: A Schema Language for XML. In *Proceedings of the 3rd Workshop on Formal Methods in Software Practice* (pp. 101–111). New York, NY, USA: ACM.

[180] Klein, M. (2002). Interpreting XML Documents via an RDF Schema Ontology. In *Proceedings of the 13th International Workshop on Database and Expert Systems Applications* (pp. 889–893).: IEEE.

[181] Klein, M., Broekstra, D., Fensel, F., van Harmelen, F., & Horrocks, I. (2003). Ontologies and Schema Languages on the Web. In D. Fensel, J. A. Hendler, H. Lieberman, & W. Wahlster (Eds.), *Spinning the Semantic Web - Bringing the World Wide Web to its Full Potential*, volume 4275 of *Lecture Notes in Computer Science* (pp. 901–908). The MIT Press, Cambridge, Massachusetts.

[182] Knublauch, H. (2009). *The SPIN Standard Modules Library*. Technical report, TopQuadrant. http://spinrdf.org/spl.html.

[183] Knublauch, H. (2011a). *SPIN - Modeling Vocabulary*. W3C Member Submission, W3C. http://www.w3.org/Submission/2011/SUBM-spin-modeling-20110222/.

[184] Knublauch, H. (2011b). *SPIN - SPARQL Syntax*. W3C Member Submission, W3C. http://www.w3.org/Submission/2011/SUBM-spin-sparql-20110222/.

[185] Knublauch, H., Hendler, J. A., & Idehen, K. (2011). *SPIN - Overview and Motivation*. W3C Member Submission, W3C. http://www.w3.org/Submission/2011/SUBM-spin-overview-20110222/.

[186] Knublauch, H. & Ryman, A. (2015a). *Shapes Constraint Language (SHACL)*. W3C First Public Working Draft, W3C. http://www.w3.org/TR/2015/WD-shacl-20151008/.

[187] Knublauch, H. & Ryman, A. (2015b). *Shapes Constraint Language (SHACL).* W3C Editor's Draft, W3C. http://w3c.github.io/data-shapes/shacl/.

[188] Kobeissy, N., Zeghlache, D., & Genet, M. G. (2007). Mapping XML to OWL for Seamless Information Retrieval in Context-Aware Environments. In *International Conference on Pervasive Services* (pp. 361–366). Istanbul, Turkey: IEEE.

[189] Kontchakov, R., Lutz, C., Toman, D., Wolter, F., & Zakharyaschev, M. (2011). The Combined Approach to Ontology-Based Data Access. In *Proceedings of the 22th International Joint Conference on Artificial Intelligence*, volume 3 (pp. 2656–2661).: AAAI Press.

[190] Kontokostas, D., Brümmer, M., Hellmann, S., Lehmann, J., & Ioannidis, L. (2014a). NLP Data Cleansing Based on Linguistic Ontology Constraints. In *Proceedings of the 11th Extended Semantic Web Conference (ESWC 2014).* http://jens-lehmann.org/files/2014/eswc_rdfunit_nlp.pdf.

[191] Kontokostas, D., Westphal, P., Auer, S., Hellmann, S., Lehmann, J., & Cornelissen, R. (2014b). Databugger: A Test-driven Framework for Debugging the Web of Data. In *Proceedings of the 23rd International Conference on World Wide Web (WWW 2014)* (pp. 115–118). New York, NY, USA: ACM. http://jens-lehmann.org/files/2014/www_demo_databugger.pdf.

[192] Kontokostas, D., Westphal, P., Auer, S., Hellmann, S., Lehmann, J., Cornelissen, R., & Zaveri, A. (2014c). Test-driven Evaluation of Linked Data Quality. In *Proceedings of the 23rd International Conference on World Wide Web (WWW 2014)* (pp. 747–758). Republic and Canton of Geneva, Switzerland. http://svn.aksw.org/papers/2014/WWW_Databugger/public.pdf.

[193] Kramer, S., Leahey, A., Southall, H., Vompras, J., & Wackerow, J. (2012). Using RDF to Describe and Link Social Science Data to Related Resources on the Web: Leveraging the Data Documentation Initiative (DDI) Model. *DDI Working Paper Series – Semantic Web, No. 1.* http://www.ddialliance.org/sites/default/files/UsingRDFToDescribeAndLinkSocialScienceDataToRelatedResourcesOnThe.Web.pdf.

[194] Kramer, S., Oechtering, A., & Wackerow, J. (2009). *Data Documentation Initiative (DDI): Entwicklung eines Metadatenstandards für Forschungsdaten in den Sozialwissenschaften.* KIM-Technology Watch Report, Kompetenzzentrum Interoperable Metadaten and DINI-Arbeitsgruppe Internationale Standardisierung in der digitalen Informationsbeschaffung (AG Standards).

[195] Kroeger, A. (2013). The Road to BIBFRAME: The Evolution of the Idea of Bibliographic Transition into a Post-MARC Future. *Cataloging & Classification Quarterly*, 51(8), 873–890.

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

[197] Kupfer, A., Eckstein, S., Störmann, B., Neumann, K., & Mathiak, B. (2007). Methods for a Synchronised Evolution of Databases and Associated Ontologies. In *Proceedings of the 2007 Conference on Databases and Information Systems IV*.

[198] Lalor, T. (2013a). *Generic Statistical Information Model (GSIM): Communication Paper for a General Statistical Audience (Version 1.1, December 2013)*. Technical report, United Nations Economic Commission for Europe (UNECE). http://www1.unece.org/stat/platform/display/gsim/GSIM+Communication+Paper.

[199] Lalor, T. (2013b). *Generic Statistical Information Model (GSIM) Specification - Version 1.1*. United Nations Economic Commission for Europe (UNECE) Specification, United Nations Economic Commission for Europe (UNECE). http://www1.unece.org/stat/platform/display/gsim/GSIM+Specification.

[200] Lausen, G., Meier, M., & Schmidt, M. (2008). SPARQLing Constraints for RDF. In *Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology (EDBT 2008)* (pp. 499–509). New York, NY, USA: ACM. http://doi.acm.org/10.1145/1353343.1353404.

[201] Layman, A., Jung, E., Maler, E., Thompson, H. S., Paoli, J., Tigue, J., Mikula, N. H., & De Rose, S. (1998). *XML-Data*. W3C Note, W3C. https://www.w3.org/TR/1998/NOTE-XML-data-0105/.

[202] Le Hors, A., Solbrig, H., & Prud'hommeaux, E. (2013). *RDF Validation Workshop Report - Practical Assurances for Quality RDF Data*. Technical report, W3C/MIT, Cambridge, MA, USA. http://www.w3.org/2012/12/rdf-val/report.

[203] Lebo, T., Sahoo, S., & McGuinness, D. (2013). *PROV-O: The PROV Ontology*. W3C Recommendation, W3C. http://www.w3.org/TR/2013/REC-prov-o-20130430/.

[204] Lee, D. & Chu, W. W. (2000). Comparative Analysis of Six XML Schema Languages. *ACM SIGMOD (Special Interest Group on Management of Data) Record*, 29(3), 76–87. http://doi.acm.org/10.1145/362084.362140.

[205] Library of Congress (2014a). *BIBFRAME Authorities*. Library of Congress Draft Specification, Library of Congress. http://www.loc.gov/bibframe/docs/bibframe-authorities.html.

[206] Library of Congress (2014b). *BIBFRAME Profiles: Introduction and Specification*. Library of Congress Draft, Library of Congress. http://www.loc.gov/bibframe/docs/bibframe-profiles.html.

[207] Library of Congress (2014c). *BIBFRAME Relationships*. Library of Congress Draft Specification, Library of Congress. http://www.loc.gov/bibframe/docs/bibframe-relationships.html.

[208] Linnerud, J., Risnes, Ø., & Gregory, A. (2015). GSIM in Practice in Norway. In *Proceedings of the Workshop on International Collaboration for Standards-Based Modernisation, United Nations Economic Com-*

mission for Europe Conference of European Statisticians Geneva, Switzerland. http://www1.unece.org/stat/platform/display/WICSBM/Geneva%2C+5-7+May+2015.

[209] Lohmann, S., Dietzold, S., Heim, P., & Heino, N. (2009). A Web Platform for Social Requirements Engineering. In J. Münch & P. Liggesmeyer (Eds.), *Software Engineering*, volume 150 of *Lecture Notes in Informatics* (pp. 309–316). Bonn, Germany: Gesellschaft für Informatik. http://subs.emis.de/LNI/Proceedings/Proceedings150/article5334.html.

[210] Lohmann, S., Heim, P., Auer, S., Dietzold, S., & Riechert, T. (2008). Semantifying Requirements Engineering - The SoftWiki Approach. In *Proceedings of the 4th International Conference on Semantic Technologies (I-SEMANTICS 2008)* (pp. 182–185). Graz, Austria.

[211] 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.

[212] Maali, F. & Erickson, J. (2014). *Data Catalog Vocabulary (DCAT)*. W3C Recommendation, W3C. http://www.w3.org/TR/2014/REC-vocab-dcat-20140116/.

[213] 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.

[214] Mechanda, K. (2015). Statistical Metadata Strategy and GSIM Implementation in Canada. In *Proceedings of the Workshop on International Collaboration for Standards-Based Modernisation, United Nations Economic Commission for Europe Conference of European Statisticians* Geneva, Switzerland. http://www1.unece.org/stat/platform/display/WICSBM/Geneva%2C+5-7+May+2015.

[215] Mellor, S. J., Kendall, S., Uhl, A., & Weise, D. (2004). *MDA Distilled*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc.

[216] Melnik, S. (1999). *Bridging the Gap between RDF and XML*. Technical report. http://infolab.stanford.edu/~melnik/rdf/fusion.html.

[217] Mernik, M., Heering, J., & Sloane, A. M. (2005). When and How to Develop Domain-Specific Languages. *ACM Computing Surveys*, 37(4), 316–344. http://doi.acm.org/10.1145/1118890.1118892.

[218] Miles, A. & Bechhofer, S. (2009a). *SKOS Simple Knowledge Organization System eXtension for Labels (SKOS-XL)*. W3C Recommendation, W3C. http://www.w3.org/TR/2009/REC-skos-reference-20090818/skos-xl.html.

[219] Miles, A. & Bechhofer, S. (2009b). *SKOS Simple Knowledge Organization System Reference*. W3C Recommendation, W3C. http://www.w3.org/TR/2009/REC-skos-reference-20090818/.

[220] Miller, E., Mueller, V., Ogbuji, U., & Baker, M. (2014). *BIBFRAME Profiles: Introduction and Specification*. Library of Congress Draft

Specification, Library of Congress. http://www.loc.gov/bibframe/docs/bibframe-profiles.html.

[221] Miller, K. & Vardigan, M. (2005). How Initiative Benefits the Research Community - the Data Documentation Initiative. In *Proceedings of the 1th International Conference on e-Social Science* Manchester, UK. http://www.ddialliance.org/sites/default/files/miller.pdf.

[222] Miller, L. (2001). *RDF Squish Query Language and Java Implementation*. Draft. http://ilrt.org/discovery/2001/02/squish/.

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

[224] Minker, J. (1982). On Indefinite Databases and the Closed World Assumption. In D. Loveland (Ed.), *6th Conference on Automated Deduction*, volume 138 of *Lecture Notes in Computer Science* (pp. 292–308). Springer Berlin Heidelberg.

[225] Minsky, M. (1974). *A Framework for Representing Knowledge*. Technical report, Cambridge, MA, USA.

[226] Moreau, L. & Missier, P. (2013). *PROV-DM: The PROV Data Model*. W3C Recommendation, W3C. http://www.w3.org/TR/2013/REC-prov-dm-20130430/.

[227] Motik, B., Grau, B. C., Horrocks, I., Wu, Z., Fokoue, A., & Lutz, C. (2012a). *OWL 2 Web Ontology Language: Profiles*. W3C Recommendation, W3C. http://www.w3.org/TR/2012/REC-owl2-profiles-20121211/.

[228] Motik, B., Horrocks, I., & Sattler, U. (2007). Adding Integrity Constraints to OWL. In *Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions*, volume 258 Innsbruck, Austria. http://ceur-ws.org/Vol-258/.

[229] Motik, B., Horrocks, I., & Sattler, U. (2009). Bridging the Gap Between OWL and Relational Databases. *Journal of Web Semantics*, 7(2), 74–89. http://www.websemanticsjournal.org/index.php/ps/article/view/159.

[230] Motik, B., Parsia, B., & Patel-Schneider, P. F. (2012b). *OWL 2 Web Ontology Language XML Serialization (Second Edition)*. W3C Recommendation, W3C. http://www.w3.org/TR/2012/REC-owl2-xml-serialization-20121211/.

[231] Murata, M., Lee, D., Mani, M., & Kawaguchi, K. (2005). Taxonomy of XML Schema Languages Using Formal Language Theory. *ACM Transactions on Internet Technology*, 5(4), 660–704.

[232] National Information Standards Organization (NISO) (2004). Understanding Metadata. *NISO Press*, (pp. 1 – 18). http://www.niso.org/publications/press/UnderstandingMetadata.pdf.

[233] Nentwich, C., Capra, L., Emmerich, W., & Finkelstein, A. (2002). Xlinkit: A Consistency Checking and Smart Link Generation Service. *ACM Transactions on Internet Technologies*, 2(2), 151–185.

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

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

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

[237] Nilsson, U. & Maluszynski, J. (1995). *Logic, Programming, and PROLOG*. New York, NY, USA: John Wiley & Sons, Inc., 2nd edition. http://dl.acm.org/citation.cfm?id=546470.

[238] Nolle, A., Meilicke, C., Stuckenschmidt, H., & Nemirovski, G. (2014). Efficient Federated Debugging of Lightweight Ontologies. In *Web Reasoning and Rule Systems* (pp. 206–215). Springer International Publishing.

[239] Nolle, A. & Nemirovski, G. (2013). ELITE: An Entailment-Based Federated Query Engine for Complete and Transparent Semantic Data Integration. In *Proceedings of the 26th International Workshop on Description Logics* (pp. 854–867).: CEUR Electronic Workshop Proceedings.

[240] Nottingham, M. and Sayre, R. (2005). *Atom Syndication Format (Atom)*. Proposed Standard Specification, IETF atompub Working Group. http://www.w3.org/2005/Atom.

[241] Obasanjo, D. (2004). *Improving XML Document Validation with Schematron*. Technical report, Microsoft Corporation. https://msdn.microsoft.com/en-us/library/aa468554.aspx.

[242] Object Management Group (OMG) (2003). *Common Warehouse Metamodel (CWM) Specification - Version 1.1*. Object Management Group (OMG) Specification, Object Management Group (OMG). http://www.omg.org/spec/CWM/1.1/.

[243] Object Management Group (OMG) (2011a). *OMG Unified Modeling Language (OMG UML), Infrastructure - Version 2.4.1*. Object Management Group (OMG) Specification, Object Management Group (OMG). http://www.omg.org/spec/UML/2.4.1/Infrastructure/.

[244] Object Management Group (OMG) (2011b). *OMG Unified Modeling Language (OMG UML), Superstructure - Version 2.4.1*. Object Management Group (OMG) Specification, Object Management Group (OMG). http://www.omg.org/spec/UML/2.4.1/Superstructure/.

[245] Object Management Group (OMG) (2012). *OMG Systems Modeling Language (OMG SysML) - Version 1.3*. Object Management Group (OMG) Specification, Object Management Group (OMG). http://www.omg.org/spec/SysML/1.3/.

[246] Object Management Group (OMG) (2013a). *Business Process Model and Notation (BPMN) - Version 2.0.2.* Object Management Group (OMG) Specification, Object Management Group (OMG). http://www.omg.org/spec/BPMN/2.0.2/.

[247] Object Management Group (OMG) (2013b). *OMG Meta Object Facility (MOF) Core Specification - Version 2.4.1.* Object Management Group (OMG) Specification, Object Management Group (OMG). http://www.omg.org/spec/MOF/2.4.1/.

[248] Object Management Group (OMG) (2014a). *Object Constraint Language - Version 2.4.* Object Management Group (OMG) Specification, Object Management Group (OMG). http://www.omg.org/spec/OCL/2.4/.

[249] Object Management Group (OMG) (2014b). *XML Metadata Interchange (XMI) Specification - Version 2.4.2.* Object Management Group (OMG) Specification, Object Management Group (OMG). http://www.omg.org/spec/XMI/2.4.2.

[250] Object Management Group (OMG) (2015). *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification - Version 1.2.* Object Management Group (OMG) Specification, Object Management Group (OMG). http://www.omg.org/spec/QVT/1.2.

[251] O'Connor, M. J. & Das, A. (2010). Semantic Reasoning with XML-Based Biomedical Information Models. *Studies in Health Technology and Informatics*, 160(2), 986–990.

[252] O'Connor, M. J. & Das, A. (2011). Acquiring OWL Ontologies from XML Documents. In *Proceedings of the 6th International Conference on Knowledge Capture (K-CAP 2011)* (pp. 17–24). New York, NY, USA: ACM.

[253] Ogbuji, C. (2000). *Validating XML with Schematron.* Technical report, O'Reilly Media, Inc. http://www.xml.com/pub/a/2000/11/22/schematron.html.

[254] Patel-Schneider, P. F. (2015). Using Description Logics for RDF Constraint Checking and Closed-World Recognition. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI-2015)* Austin Texas, USA. http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9531.

[255] Pérez, J., Arenas, M., & Gutierrez, C. (2009). Semantics and Complexity of SPARQL. *ACM Transactions on Database Systems (TODS)*, 34(3), 16:1–16:45.

[256] Pohl, K. (2010). *Requirements Engineering - Fundamentals, Principles, and Techniques.* Springer-Verlag Berlin Heidelberg. https://www.springer.com/us/book/9783642125775.

[257] Polleres, A., Hogan, A., Delbru, R., & Umbrich, J. (2013). RDFS and OWL Reasoning for Linked Data. In S. Rudolph, G. Gottlob, I. Horrocks, & F. van Harmelen (Eds.), *Reasoning Web. Semantic Technologies for Intelligent Data Access*, volume 8067 of *Lecture Notes in Computer Science* (pp. 91–149). Springer Berlin Heidelberg.

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

[259] PREMIS Editorial Committee (2015). *PREMIS Data Dictionary for Preservation Metadata - Version 3.0*. PREMIS Specification, PREMIS Editorial Committee. http://www.loc.gov/standards/premis/v3/premis-3-0-final.pdf.

[260] Prud'hommeaux, E. (2014). *Shape Expressions 1.0 Primer*. W3C Member Submission, W3C. http://www.w3.org/Submission/2014/SUBM-shex-primer-20140602/.

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

[262] Prud'hommeaux, E. & Carothers, G. (2014). *RDF 1.1 Turtle - Terse RDF Triple Language*. W3C Recommendation, W3C. http://www.w3.org/TR/2014/REC-turtle-20140225/.

[263] Prud'hommeaux, E., Labra Gayo, J. E., & Solbrig, H. (2014). Shape Expressions: An RDF Validation and Transformation Language. In *Proceedings of the 10th International Conference on Semantic Systems (SEMANTiCS)*, SEMANTiCS 2014 (pp. 32–40). New York, NY, USA: ACM.

[264] Raja, A. & Lakshmanan, D. (2010). Domain Specific Languages. *International Journal of Computer Applications (IJCA)*, 1(21), 99–105. http://www.ijcaonline.org/journal/number21/pxc387640.pdf.

[265] Rasmussen, K. B. (2013). Social Science Metadata and the Foundations of the DDI. *IASSIST Quarterly*, 37(1), 28–35. http://iassistdata.org/sites/default/files/iq/iqvol371_4_rasmussen.pdf.

[266] Reif, G., Gall, H., & Jazayeri, M. (2005). WEESA: Web Engineering for Semantic Web Applications. In *Proceedings of the 14th International Conference on World Wide Web* (pp. 722–729). New York, NY, USA: ACM.

[267] Reiter, R. (1978). On Closed World Data Bases. In H. Gallaire & J. Minker (Eds.), *Logic and Data Bases* (pp. 55–76). Springer US.

[268] 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.

[269] Reynolds, D. (2014). *The Organization Ontology*. W3C Recommendation, W3C. http://www.w3.org/TR/2014/REC-vocab-org-20140116/.

[270] Robie, J., Chamberlin, D., Dyck, M., & Snelson, J. (2014a). *XML Path Language (XPath) 3.0*. W3C Recommendation, W3C. http://www.w3.org/TR/2014/REC-xpath-30-20140408/.

[271] Robie, J., Chamberlin, D., Dyck, M., & Snelson, J. (2014b). *XQuery 3.0: An XML Query Language*. W3C Recommendation, W3C. http://www.w3.org/TR/2014/REC-xquery-30-20140408/.

[272] Rudolph, S. (2011). Foundations of Description Logics. In A. Polleres, C. d'Amato, M. Arenas, S. Handschuh, P. Kroner, S. Ossowski, & P. Patel-Schneider (Eds.), *Reasoning Web. Semantic Technologies for the Web of Data*, volume 6848 of *Lecture Notes in Computer Science* (pp. 76–136). Springer Berlin Heidelberg.

[273] Russell, S. & Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ, USA: Prentice Hall Press, 3rd edition.

[274] Ryman, A. (2014). *Resource Shape 2.0*. W3C Member Submission, W3C. http://www.w3.org/Submission/2014/SUBM-shapes-20140211/.

[275] Ryman, A. G., Hors, A. L., & Speicher, S. (2013). OSLC Resource Shape: A Language for Defining Constraints on Linked Data. In C. Bizer, T. Heath, T. Berners-Lee, M. Hausenblas, & S. Auer (Eds.), *Proceedings of the International World Wide Web Conference (WWW), Workshop on Linked Data on the Web (LDOW)*, volume 996 of *CEUR Workshop Proceedings*. http://ceur-ws.org/Vol-996/.

[276] Scanu, M. (2015). GSIM Implementation in the Istat Metadata System. In *Proceedings of the Workshop on International Collaboration for Standards-Based Modernisation, United Nations Economic Commission for Europe Conference of European Statisticians* Geneva, Switzerland. http://www1.unece.org/stat/platform/display/WICSBM/Geneva%2C+5-7+May+2015.

[277] 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.

[278] Schmachtenberg, M., Bizer, C., & Paulheim, H. (2014). *State of the LOD Cloud 2014*. Technical report, University of Mannheim. http://linkeddatacatalog.dws.informatik.uni-mannheim.de/state/.

[279] 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/.

[280] Schreiber, G. & Raimond, Y. (2014). *RDF 1.1 Primer*. W3C Working Group Note, W3C. http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/.

[281] Sáenz-Pérez, F., Caballero, R., & García-Ruiz, Y. (2011). A Deductive Database with Datalog and SQL Query Languages. In H. Yang (Ed.), *Programming Languages and Systems*, volume 7078 of *Lecture Notes in Computer Science* (pp. 66–73). Springer Berlin Heidelberg. http://dx.doi.org/10.1007/978-3-642-25318-8_8.

[282] Shabo, A., Rabinovici-Cohen, S., & Vortman, P. (2006). Revolutionary Impact of XML on Biomedical Information Interoperability. *IBM Systems Journal*, 45(2), 361–372.

[283] Shadbolt, N., Berners-Lee, T., & Hall, W. (2006). The Semantic Web Revisited. *IEEE Intelligent Systems*, 21(3), 96–101. http://www.computer.org/csdl/mags/ex/2006/03/x3096-abs.html.

[284] Simister, S. & Brickley, D. (2013). Simple Application-Specific Constraints for RDF Models. In *W3C RDF Validation Workshop. Practical Assurances for Quality RDF Data* Cambridge, MA, USA. http://www.w3.org/2012/12/rdf-val/.

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

[286] Sirin, E. & Tao, J. (2009). Towards Integrity Constraints in OWL. In *Proceedings of the 6th International Workshop on OWL: Experiences and Directions (OWLED 2009), 8th International Semantic Web Conference (ISWC 2009)* Chantilly, Virginia, USA. http://webont.org/owled/2009/.

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

[288] Solbrig, H. & Prud'hommeaux, E. (2015). *A Formal Model of the Shape Expression Language*. Technical report. https://github.com/w3c/ShEx/blob/master/ShExZ/ShExZ.pdf.

[289] Sporny, M., Kellogg, G., & Lanthaler, M. (2014). *JSON-LD 1.0 - A JSON-based Serialization for Linked Data*. W3C Recommendation, W3C. http://www.w3.org/TR/2014/REC-json-ld-20140116/.

[290] Statistical Data and Metadata eXchange (SDMX) (2013). *SDMX 2.1 Technical Specification – Consolidated Version 2013*. SDMX Technical Specification, Statistical Data and Metadata eXchange (SDMX). http://sdmx.org/?page_id=10.

[291] Statistical Data and Metadata eXchange (SDMX) (2014). *SDMX Statistical Guidelines - Content-Oriented Guidelines*. SDMX Technical Specification, Statistical Data and Metadata eXchange (SDMX). http://sdmx.org/?page_id=11.

[292] Staworko, S., Boneva, I., Gayo, J. E. L., Hym, S., Prud'hommeaux, E. G., & Solbrig, H. (2015). Complexity and Expressiveness of ShEx for RDF. In M. Arenas & M. Ugarte (Eds.), *18th International Conference on Database Theory (ICDT 2015)*, volume 31 of *Leibniz International Proceedings in Informatics (LIPIcs)* (pp. 195–211). Dagstuhl, Germany: Schloss Dagstuhl –Leibniz Center for Informatics.

[293] Steinberg, D., Budinsky, F., Paternostro, M., & Merks, E. (2009). *EMF: Eclipse Modeling Framework*. Addison-Wesley Professional, 2nd edition.

[294] Stuckenschmidt, H. (2011). *Ontologien, Konzepte, Technologien und Anwendungen*. Berlin, Heidelberg, Germany: Springer.

[295] Taha, W. (2008). Domain-Specific Languages. In *Proceeding of the 15th International Conference on Computer Engineering and Systems (ICCES 2008)* Houston, TX, USA. http://www.cs.rice.edu/~taha/publications/conference/icces08.pdf.

[296] Tandy, J., Ceolin, D., & Stephan, E. (2016). *CSV on the Web: Use Cases and Requirements*. W3C Working Group Note, W3C. https://www.w3.org/TR/2016/NOTE-csvw-ucr-20160225/.

[297] Tandy, J. & Herman, I. (2015). *Generating JSON from Tabular Data on the Web*. W3C Recommendation, W3C. https://www.w3.org/TR/2015/REC-csv2json-20151217/.

[298] Tandy, J., Herman, I., & Kellogg, G. (2015). *Generating RDF from Tabular Data on the Web*. W3C Recommendation, W3C. https://www.w3.org/TR/2015/REC-csv2rdf-20151217/.

[299] Tao, J. (2012). *Integrity Constraints for the Semantic Web: An OWL 2 DL Extension*. PhD thesis, Rensselaer Polytechnic Institute.

[300] Tao, J., Sirin, E., Bao, J., & McGuinness, D. L. (2010). Integrity Constraints in OWL. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010)* Atlanta, Georgia, USA. https://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/viewFile/1931/2229.

[301] Tennison, J. (2016). *CSV on the Web: A Primer*. W3C Working Group Note, W3C. https://www.w3.org/TR/2016/NOTE-tabular-data-primer-20160225/.

[302] Tennison, J. & Kellogg, G. (2015a). *Metadata Vocabulary for Tabular Data*. W3C Recommendation, W3C. https://www.w3.org/TR/2015/REC-tabular-metadata-20151217/.

[303] Tennison, J. & Kellogg, G. (2015b). *Model for Tabular Data and Metadata on the Web*. W3C Recommendation, W3C. https://www.w3.org/TR/2015/REC-tabular-data-model-20151217/.

[304] Thompson, H. S., Beech, D., Maloney, M., & Mendelsohn, N. (2004). *XML Schema Part 1: Structures Second Edition*. W3C Recommendation, W3C. http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/.

[305] Tous, R., García, R., Rodríguez, E., & Delgado, J. (2005). Architecture of a Semantic XPath Processor. Application to Digital Rights Management. In K. Bauknecht, B. Pröll, & H. Werthner (Eds.), *E-Commerce and Web Technologies*, volume 3590 of *Lecture Notes in Computer Science* (pp. 1–10). Springer Berlin Heidelberg.

[306] Tsarkov, D. & Horrocks, I. (2006). FaCT++ Description Logic Reasoner: System Description. In *Automated Reasoning* (pp. 292–297). Springer.

[307] Vale, S. (2010). Exploring the Relationship between DDI, SDMX and the Generic Statistical Business Process Model. *DDI Working Paper Series*. http://www.ddialliance.org/resources/publications/working/othertopics/ExploringRelationshipBetweenDDI-SDMX-GSBPM.pdf.

[308] van der Vlist, E. (2001). *Comparing XML Schema Languages*. Technical report, O'Reilly Media, Inc. http://www.xml.com/pub/a/2001/12/12/schemacompare.html.

[309] van der Vlist, E. (2002a). *Relax NG, Compared*. Technical report, O'Reilly Media, Inc. http://www.xml.com/lpt/a/2002/01/23/relaxng.html.

[310] van der Vlist, E. (2002b). *XML Schema - The W3C's Object-Oriented Descriptions for XML*. O'Reilly Media, Inc. http://shop.oreilly.com/product/9780596002527.do.

[311] van der Vlist, E. (2003). *RELAX NG*. O'Reilly Media, Inc. http://shop.oreilly.com/product/9780596004217.do.

[312] van der Vlist, E. (2007). *Schematron*. O'Reilly Media, Inc. https://library.oreilly.com/book/9780596527716/schematron/toc.

[313] van der Vlist, Eric (2013). *Examplotron*. Specification, Dyomedea. http://examplotron.org/0/8/.

[314] Vardigan, M. (2013a). DDI Timeline. *IASSIST Quarterly*, 37(1), 51–56. http://iassistdata.org/downloads/iqvol371_4_vardigan2.pdf.

[315] Vardigan, M. (2013b). The DDI Matures: 1997 to the Present. *IASSIST Quarterly*, 37(1), 45–50. http://iassistdata.org/downloads/iqvol371_4_vardigan.pdf.

[316] Vardigan, M., Heus, P., & Thomas, W. (2008). Data Documentation Initiative: Toward a Standard for the Social Sciences. *International Journal of Digital Curation*, 3(1), 107 – 113. http://www.ijdc.net/index.php/ijdc/article/view/66.

[317] Volz, R., Oberle, D., Staab, S., & R., S. (2003). *OntoLiFT Prototype – WonderWeb: Ontology Infrastructure for the Semantic Web*. Technical report. http://wonderweb.man.ac.uk/deliverables/documents/D11.pdf.

[318] 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.

[319] Vrandecic, D., Lange, C., Hausenblas, M., Bao, J., & Ding, L. (2010). Semantics of Governmental Statistics Data. In *Proceedings of the 2nd International ACM Web Science Conference (WebSci 2010)* Raleigh, North Carolina, USA. http://journal.webscience.org/400/2/websci10_submission_118.pdf.

[320] Wackerow, J. (2008). The Data Documentation Initiative (DDI). In *Proceedings of the 8th DCMI International Conference on Dublin Core and Metadata Applications (DC 2008)* Berlin, Germany.

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

[322] Walmsley, P. (2012). *Definitive XML Schema*. Upper Saddle River, NJ, USA: Prentice Hall Press, 2nd edition. http://dl.acm.org/citation.cfm?id=2392683.

[323] Wilson, Tim (2008). *KML - Version 2.2.0*. Open Geospatial Consortium Inc. Specification, Open Geospatial Consortium Inc. http://www.opengeospatial.org/standards/kml.

[324] Wira-Alam, A. & Hopt, O. (2009). Implementing DDI 3: The German Microcensus Case Study. *IASSIST Quarterly*, 33(1), 16–22. http://www.iassistdata.org/sites/default/files/iq/iqvol3312hopt.pdf.

[325] Wyke, R. A. & Watt, A. (2002). *XML Schema Essentials*. New York, NY, USA: John Wiley & Sons, Inc.

Erklärung

(gemäß §4, Abs. 4 der Promotionsordnung vom 15. August 2006)

Ich versichere wahrheitsgemäß, die Dissertation bis auf die in der Abhandlung angegebene Hilfe selbständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und genau kenntlich gemacht zu haben, was aus Arbeiten anderer und aus eigenen Veröffentlichungen unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, den *08.07.2016*