Towards maintainable constraint validation and repair for taxonomies: The PoolParty approach

Christian Mader[1] and Monika Solanki[2]

[1] Semantic Web Company, Austria
c.mader@semantic-web.at
[2] Department of Computer Science, University of Oxford, UK
monika.solanki@cs.ox.ac.uk

Abstract. The specification and validation of constraints in semantically annotated datasets has been receiving increasing attention in the last few years. In this paper we present requirements for using such approaches in an industrial setting. We underpin these requirements with concrete examples of data constraints, as encountered during the development and operation of the PoolParty Thesaurus Server. We show how these constraints can be expressed in SPARQL and propose a SHACL-based approach that combines data validation specification with repair strategies. Based on an implementation that is driven by practical use cases, we show that this approach is able to validate and repair datasets that are provided by customers of PoolParty as well as those openly available on the Web.

1 Introduction

Semantic Web Company[3] (SWC) provides one of the leading commercial taxonomy management applications, the PoolParty Thesaurus Server (PPT). In recent years, it has evolved into an authoring tool for knowledge graphs that make use of standard schemas such as SKOS[4], DCTERMS[5], or FOAF[6].

Taxonomists use PPT to integrate a variety of models, schemata, ontologies and vocabularies into their knowledge bases. From a data-centric perspective, the taxonomy import functionality is one of the most crucial utilities provided in PPT, to interact with third-party datasets. PoolParty supports simple import of existing lists, spreadsheets or taxonomies residing on a local drive or on the Web. This leverages the value of existing taxonomies and makes knowledge organization systems part of the Semantic Web.

One of the challenges in combining varied data sources is to ensure that these data mashups at any time conform to a set of quality heuristics. This is needed because applications such as PPT, that consume data from external sources

[3] https://www.semantic-web.at/
[4] http://www.w3.org/2004/02/skos/
[5] http://dublincore.org/documents/dcmi-terms/
[6] http://www.foaf-project.org/

including the Web, rely on data processing algorithms which expect a defined set of input data. For instance, PPT accesses DBpedia subject categories whose hierarchical structure frequently changes so that it violates the requirements of Poolparty's internal data model. Furthermore, as taxonomies transist from "simple" thesauri to fully-fledged ontologies with rich semantics, users of PPT want to impose custom semantics and restrictions on the datasets they develop.

Validating data against a set of data-driven rules or constraints[7] that ensure the desired data quality is therefore a critical task that needs to be made a part of any workflow involving PPT, both from an internal developer and an external stakeholder perspective. Keeping validation rules or data consistency constraints aligned with changes in PPT's data processing logic requires additional effort in terms of development resources, so data constraints must be easily maintainable.

Good quality data in combination with means for easy data constraint management can therefore (1) ensure that Linked Data consuming applications work as expected, (2) make sure the datasets which customers develop fulfil the intended usage scenarios, and (3) relieve SWC of some of the additional overheads in terms of resource allocation for data-related debugging issues. In this paper, we present the results of an evaluation carried out to identify the constraint violations that typically occur when datasets are imported into PPT. We focus on (semi-) automated detection and repair of constraint violations (e.g., missing or inconsistent data) in datasets, both imported as well as developed within PPT.

The paper is structured as follows: Section 3 presents the requirements analysis undertaken to assess the need for constraints validation. Section 4 discusses our dataset selection methodology. Section 5 provides an overview of the formalisation of constraints. Section 6 presents our constraint validation and repair techniques. Section 7 presents our evaluation methodology and discusses the results of our evaluation. Section 2 discusses related work and, finally, Section 8 presents our conclusions and future work.

2 Related Work

RuleML[8] is a family of rule languages serialised in XML. RuleML [2] covers a wide spectrum of rules from deliberation, derivation, transformation and reaction rules. SWRL[9] is an expressive rule language that can be used to increase the amount of knowledge encoded in OWL ontologies. RIF[10] is a W3C standard designed for interchanging rules between different rule systems. OWL-RL[11] is a syntactic subset of OWL 2 which is amenable to implementation using rule-

[7] In this paper, rules and constraints are used interchangeably.
[8] `http://www.ruleml.org/`
[9] `http://www.w3.org/Submission/SWRL/`
[10] `http://www.w3.org/TR/rif-overview/`
[11] `http://www.w3.org/TR/owl2-profiles/#OWL_2_RL`

based technologies. Provision for the specification of rules have also been made in Semantic Web frameworks and triple stores (e.g., Jena[12] or OWLIM[13]).

The use of SPARQL as a rule language based on its `CONSTRUCT` keyword has long been advocated [9]. Specialised extensions of SPARQL for the rule based processing of events such as EP-SPARQL [1] have also been proposed. Another approach that exploits SPARQL for data validation is RDFUnit [6] which uses a test-driven methodology usable for either direct application on Linked Datasets, or indirectly by evaluating the quality of mapping files that in turn are used to generate Linked Datasets [3].

A pragmatic approach towards dataset analysis is taken by qSKOS [7], which is a tool for evaluating potential quality problems („quality issues") in datasets that use SKOS. SPIN (SPARQL Inferencing notation) is a SPARQL-based rule and constraint language for the Semantic Web. SPIN links class definitions with SPARQL queries using the SPIN modelling vocabulary, to capture constraints and rules. The use of SPARQL and SPIN to identify data quality problems on the Semantic Web has been proposed in [4].

SHACL[14](Shapes constraint language) is a specification under development by the W3C RDF Data shapes working group. We highlight the advantages of SHACL as used in our proposed approach in Section 5.

Hansen et al. [5] introduce Validata, an online tool that tests conformance of RDF datasets against specifications written in the Shape Expression language (ShEx) [10]. In this paper we follow a similar approach, but build on SHACL for constraint specification. In contrast to ShEx, SHACL allows constraint definitions to be specified as RDF datasets, thus levering the potential of Linked Data such as easy reuse and extension by, e.g., attaching repair strategies as we have suggested in this paper.

Work on repairing datasets that use the SKOS schema has been done by Suominen et al. [11, 12]. They introduce Skosify, a tool that can automatically resolve potential quality issues such as hierarchical circularities or overlapping labels. The resolution logic is built into Skosify and changes to the code must be made if treatment of the found issues should be adapted or extended.

While the above reported approaches can be used to implement validation of RDF datasets, most of them require the creation of sophisticated rule bases to check against each quality dimension separately. This is hard to maintain and require additional knowledge of the rule language syntax (ShEx) or templating capabilites (RDFUnit). We therefore focus on an approach using SHACL because of its capability to abstract from application of basic rules in favour of a higher level description of a graph's structure.

---

[12] `http://jena.sourceforge.net/inference/#rules`

[13] `https://confluence.ontotext.com/display/OWLIMv43/OWLIM-SE+Reasoner#OWLIM-SEReasoner-RuleBasedInference`

[14] `http://www.w3.org/TR/shacl/`

3 Requirements Analysis

As part of our requirements analysis, we identified several areas in PPT where constraint violations could potentially occur:

3.1 Data Consistency Constraints

PPT uses a triple-store to persist taxonomy information. Changes to these datasets are currently performed in two different ways: First, by executing atomic "actions", which encapsulate triple changes (i.e., additions and removals). Currently, checks if an action contains valid triples in a way that it does not violate the store's data consistency are scattered in the code and sometimes performed multiple times, making them hard to maintain. There is also a "legacy way" by adding and deleting triples directly from the store with no constraint validation at all. We therefore need a way to

R1 Provide a mechanism to specify data constraints in a formal way, fit for usage with the various PPT software components as well as easily understandable and changeable also by end-users.

PPT components impose constraints on the data they process. It is most critical that the core PPT constraints are met at any time, otherwise the application may fail. Therefore, in this paper, most of the exemplary constraints cover these core constraints. Currently PPT fails to display, or, in the worst case, produces error messages if it encounters data not conforming to these constraints. SWC works towards a constraint-based system which would be able to tell the user why data validation has failed and suggest ways in which the user can address the problem(s). This results in the requirements:

R2 Identify and analyse customer datasets that are imported into PPT and are a source of constraint violations.

R3 Provide a validation mechanism to check for constraint violation and evaluate this against the selected datasets.

3.2 Requirements for Constraint Resolution

In the semantic space, constraint violations can be extremely difficult to understand. This is because they may come from a number of different sources or are the result of combining data from these sources – for example when merging two ontologies as it is, e.g., done when importing datasets. Users of PPT would greatly benefit from tools which could help them to understand more easily what the cause of constraint conflicts is and what pathways are available to resolve them. We need a method for formulating actions for responding to constraint violations in a generic way, leading to requirement

R4 Combine formal data constraint definitions with reusable repair strategies that can be easily applied by end-users in a (semi-) automatic way.

4 Dataset and Constraint Selection

In order to investigate the extent at which consistency constraint violations constitute a problem, we performed an analysis on a selection of datasets created by SWC, customers of SWC, and publicly available on the Web. We complement the datasets by a selection of consistency constraints. In this section we address requirement **R2**.

Regarding dataset selection, SWC maintains a list of datasets that are requested by customers for use with PPT either as the basis of a new taxonomy or as target vocabularies for mapping and alignment. For some of these datasets, SWC implemented methods[15] to convert them into taxonomies that make use of the SKOS schema and are suitable for import into PPT. The list of datasets also contains taxonomies that have been created by third parties using undocumented conversion methods. Furthermore, SWC also identified datasets for which it is currently unknown if they can be used with PPT without conversion. We reviewed the list of datasets and assigned each dataset to one of the following groups: *SWC-generated* - Datasets for which a conversion to a PPT-compatible taxonomy has been performed by SWC (containing 10 datasets), *Custom-generated*: Datasets for which a conversion to a PPT-compatible taxonomy has been performed by third-party institutions (containing 9 datasets), and *Web*: Datasets that are using SKOS, but for which is currently unknown if they are compatible with PPT (containing 7 datasets).

Overall, this procedure led to identification of 26 datasets, which we downloaded either from SWC-internal file servers or, if no version prepared for PPT existed, from the dataset website.

Regarding the selection of constraints, we identified 16 data validation constraints which must be fulfilled by a dataset so that it can be used properly with PPT. There is currently no well-defined application schema, however, we infered these constraints from the application logic and from experiences with importing customer-provided datasets into PPT. We selected six constraints which only address the parts of the dataset that use elements of SKOS. Other validation constraints required by different PPT components also rely on project metadata, such as the default language of the project or the projects linked to the taxonomy. However, these are subject of our future work.

5 Constraint Specification

In this section we address requirements **R1** and **R4** by (i) specifying the constraints we identified through the method described in Section 4 and (ii) introducing an approach to interweave validation constraints with repair strategies.

[15] using, e.g., script languages such as Perl or altering the RDF graph of the original vocabulary with SPARQL.

5.1 Data Validation Constraints

We present exemplary data consistency constraints which should be met in order for PPT to function as expected. Some of these constraints have already been defined [6–8], some are specific to PPT.

BidirectionalRelationsHierarchical (br): PPT's algorithms for taxonomy display and processing rely on having both directions of a hierarchical relation materialized as triples. This constraint is also contained in the catalogue of checks performed by RDFUnit [6].

```
SELECT DISTINCT ?resource WHERE {
 ?someRes ?p ?resource.
 ?p owl:inverseOf ?pInv.
 FILTER NOT EXISTS {?resource ?pInv ?someRes}}
```

ConceptTypeAssertion (cta): PPT expects that for each concept, membership to the `skos:Concept` class is explicitly asserted because no RDFS inferencing is performed by default.

```
SELECT DISTINCT ?resource WHERE {
 ?resource skos:broader|skos:narrower ?otherRes.
 FILTER NOT EXISTS {?resource a skos:Concept}}
```

HierarchicalConsistency (hc): PPT manages taxonomies as tree-like graphs where each concept must have a parent resource.

```
SELECT DISTINCT ?resource WHERE {
 ?resource a skos:Concept
 FILTER NOT EXISTS {
  ?resource (skos:broader|^skos:narrower)*/skos:topConceptOf ?parent.
  ?parent a skos:ConceptScheme.}}
```

LabelAmbiguities (lam): Distinct resources labelled identically may constitute a quality problem (e.g., duplicate or redundant concepts). This constraint is not mandatory to be met by the data but hint at improving the taxonomy [7].

```
SELECT DISTINCT ?resource WHERE {
 ?resource ?labelProp1 ?label.
 ?otherRes ?labelProp2 ?label.
 FILTER (?labelProp1 IN (skos:prefLabel,skos:altLabel,skos:hiddenLabel))
 FILTER (?labelProp2 IN (skos:prefLabel,skos:altLabel,skos:hiddenLabel))
 FILTER (?resource != ?otherRes)}
```

UniquePreferredLabels (upl): As pointed out by the SKOS reference documentation [8], a concept must not have assigned two distinct preferred labels in the same language. Since PPT closely follows the SKOS specification, this constraint must be met in order to ensure data consistency.

```
SELECT DISTINCT ?resource {
 ?resource skos:prefLabel ?pl1.
 ?resource skos:prefLabel ?pl2.
 FILTER ((?pl1 != ?pl2) && (LANG(?pl1) = LANG(?pl2)))}
```

5.2 Validation using SHACL

Having one SPARQL query for every constraint is hard to read and maintain. In order to overcome these limitations we specify our constraints encapsulated within SHACL declarations as it provides the following advantages over direct SPARQL representation.

- *Abstraction*: We use SHACL as a kind of "abstraction layer" over SPARQL as we run the validations directly over the data in our triple store.
- *Declarative*: SHACL provides a nice declarative mechanism to associate constraints with nodes in the data graph that have various scoping options.
- *Composition*: SHACL provides a way to define a kind of "specification" an RDF graph has to be conformant to. It is therefore possible to address multiple consistency constraints in one RDF document.

As highlighted earlier in Section 2, it is worth noting that SPARQL and SHACL are meant to complement rather than compete with each other. The following example shows how the constraint *upl* can be expressed in SHACL.

```
ppts:ConceptShape
 a sh:Shape;
 sh:scopeClass skos:Concept;
 sh:property [
  a sh:PropertyConstraint;
  sh:predicate skos:prefLabel;
  sh:minCount 1;
  sh:minLength 1;
  sh:datatype rdf:langString;
  sh:uniqueLang true].
```

5.3 Integration of Repair Strategies

To interweave validation constraints with repair strategies, we developed a basic vocabulary. Below, we show an exemplary shape, expressing the constraint which says that resources having assigned another resource by the `skos:broader` property, must also have an incoming link asserted by the `skos:narrower` property. If this constraint is violated, the repair strategy `rs:AddInverseStrategy` should be executed, which adds this missing reciprocal relation. We provide details on the implementation in Section 6.

```
ppts:ConceptHavingBroader
 a sh:Shape;
 sh:scope [
  a sh:Scope;
  a sh:PropertyScope ;
  sh:predicate skos:broader];
 sh:inverseProperty [
  a sh:InversePropertyConstraint;
  sh:predicate skos:narrower;
  sh:minCount 1;

  rs:strategy [
    a rs:AddInverseStrategy]].
```

The type of the repair strategy defined in the example above refers to a resolution algorithm built into our framework. This is just a convention, as repair strategies can, e.g., also be stated as SPARQL `INSERT` operations that are parameterized and executed.

6 Implementation

In this section we address requirement **R3**. To detect the datasets that violate the data validation constraints introduced in Section 5.1, we implemented[16] a Java application that takes as an input the dataset file and outputs a report containing the resources that cause the violation of a specific constraint. Constraint validation is done in multiple steps: First, a Sesame[17] in-memory repository is created. It is initialized with both the SKOS data model[18] as well as the dataset that is being validated. For each constraint, the implementation executes the corresponding SPARQL query and assembles the violation report.

Our implementation for repairing datasets is based on SHACL and the implementation provided by Holger Knublauch[19]. We created a Java tool which takes as an input the dataset that should be validated and repaired as well as its constraints formalized by following the SHACL specification[20]. The tool then creates a Jena[21] RDF model containing this dataset, the shapes formalization and the SHACL schema. Based on these, validation is performed and a RDF report containing the validation errors is created.

For dataset repair, we again create an in-memory Sesame repository containing the validation report, the SKOS schema, the dataset's constraints (in SHACL) and the dataset itself. Based on the set of constraint violations, our implementation instantiates the repair strategies that have been specified in the provided constraint definitions. Each repair strategy returns a triple changeset, i.e., sets of RDF statements that should be added or removed. This changeset is then applied to the repository and the dataset can be considered repaired.

Our methodology for identifying and repairing constraint violations has an impact on a multitude of Linked Data consuming and processing applications. They must make sure that the data it processes must meet certain requirements imposed by the closed-world environment, i.e., their business logic. Since the technologies we use in our implementation are based on SPARQL, scalability for constraint checking largely depends on the effectiveness of the used RDF store. When using SHACL for constraint specification, time complexity depends on the effectiveness of the SHACL implementation, which, according to the SHACL specification, is not bound to SPARQL.

---

[16] `https://github.com/cmader/dataconsistency`
[17] `http://rdf4j.org/`
[18] retrieved from `http://www.w3.org/2009/08/skos-reference/skos.rdf`
[19] `https://github.com/TopQuadrant/shacl`
[20] `http://w3c.github.io/data-shapes/shacl/`
[21] `http://jena.apache.org/documentation/rdf/`

7 Evaluation

In this section, we provide results of our implementation of the SPARQL-based constraint violation introduced in Section 5.1 as well as a SHACL-based constraint validation using one exemplary constraint definition and repair strategy (introduced in Section 5.3).

7.1 SPARQL-based Dataset Validation Results

Table 1 shows an overview of the observed constraint violations in the datasets we identified. For each data consistency constraint we provide the number of datasets that violate this constraint.

As expected, violations of the constraint *cta* never occurred in datasets that were converted into PPT taxonomies. Interestingly, violations of *br*, *hc*, and *lam* occur also in these converted taxonomies. The reason for this is that the datasets have been extracted from earlier versions of PPT or were created using conversion scripts designed for these earlier versions. Only one constraint, *upl*, was never observed in any of the datasets. The reason might be that *upl* is not a PPT-specific constraint but specified in the SKOS reference documentation and is therefore respected by most dataset providers.

We can observe that both datasets being converted for PPT as well as datasets on the Web violate consistency constraints. Therefore, during the development process of an application, it is crucial to maintain and evolve data consistency constraint definitions in the same way as the application's code evolves. The data (constraints) and software development lifecycles must be aligned to each other and there is a need for approaches, such as the one proposed in this paper, that aim to support this goal.

Table 1. Number of datasets violating a specific constraint.

Constraint	**SWC-generated**	**Custom-generated**	**Web**
ConceptTypeAssertion (cta)	0	0	2
BidirectionalRelationsHierarchical (br)	6	5	6
HierarchicalConsistency (hc)	4	3	5
DctermsCreatorLiteral (dcl)	0	0	1
LabelAmbiguities (lam)	8	9	4
UniquePreferredLabels (upl)	0	0	0

Figure 1 illustrates how the constraints we expressed as SPARQL queries (Section 5.1) perform in relation to the number of statements they contain. To maintain readability, we omitted 10 datasets that contain less than 50,000 statements. For each of the remaining datasets, we computed the arithmetic mean of the time the six constraint checks take. We can observe that there is no correlation between a dataset's size and the time needed to perform the constraint

validation. The difference in validation runtime, however, is determined by the structure of the dataset. The constraint *lam*, for instance, compares different combinations of labels, of multiple concepts, depending on their type (preferred, alternative, or hidden label). If some concepts lack one or more of these types, the number of combinations drops significantly, resulting in a lower validation time. This situation applies in a similar way also to other constraints.

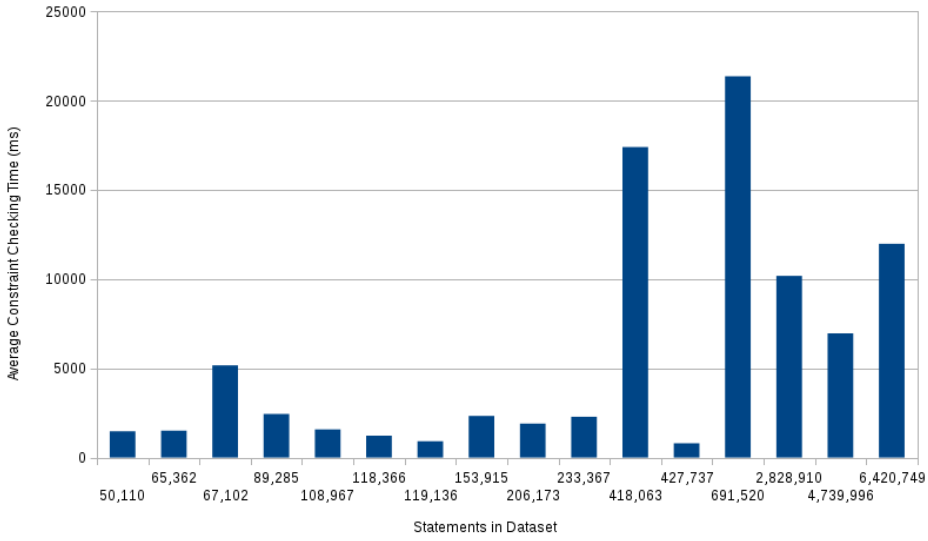

Fig. 1. Dataset Validation Performance

7.2 Dataset Repair Results using SHACL

To find out how our approach performs in a practical setting, we implemented a repair strategy and applied it on the datasets we retrieved from the Web. The repair strategy we chose is a special case of the constraint *br* which we introduced in see Section 5.1. It complements the relation `skos:broader` for two resources R_1 and R_2 if the dataset has asserted `skos:narrower` for R_2 and R_1 and vice versa. Parts of the SHACL definition of the according data shape have been described in Section 5.3.

Table 2 lists the selected datasets, ordered by the number of statements they contain. For each dataset we also list the time loading it into the memory store for validation as well as the time needed for validation against the SHACL definitions. For those datasets which failed validation, we provide the time needed for setting up the repair environment (Repair Setup Time) and the time needed for performing the actual repair (Repair Time), i.e., creating the required RDF triple changeset. We also provide the number of statements that were automatically added by the implemented repair strategy.

The reason why, compared to Table 1, fewer datasets are affected by the constraint (no added statements) is that the validation and repair constraint only takes into account the two relations `skos:broader` and `skos:narrower` and not other relations being defined as `owl:inverseOf` each other (Section 5.1).

Our results indicate that SHACL-based constraint verification and repair is feasible for datasets on the Web. Our current implementation is not optimal in terms of runtime performance as it uses different RDF frameworks causing overhead in tooling setup. In an optimized version, we will therefore be able to eliminate the time for setting up the repair environment. However, this evaluation showed us that dataset validation with the used SHACL implementation performs and scales well even for larger datasets.

Table 2. Repair strategy execution performance.

Dataset	Total Statements	Loading Time (ms)	Validation Time (ms)	Repair Setup Time (ms)	Repair Time (ms)	Added Statements
gemet-skoscore.rdf	32596	5192	11986	0	0	0
stw.ttl	108967	5923	14735	0	0	0
psh-skos.rdf	119136	9052	18762	0	0	0
thesoz_0_93.rdf	427737	21654	14625	12576	283	1
eurovoc_skos.rdf	2828910	112836	23366	60520	35112	6922
agrovoc_2015	4739996	295628	43316	179526	567247	33656
esco_skos.rdf	6420749	245587	20277	0	0	0

8 Conclusions

In this paper we have presented requirements towards establishing data consistency management in an industrial setting for the PPT application. We introduced examples of data consistency constraints and showed how these constraints can be formalized using SPARQL. We also demonstrated how SHACL can be used to express them in a more declarative and concise way. Furthermore, we introduced an approach how to interweave SHACL-based data consistency specification with repair strategies and provided information how such an approach performs when used for repairing datasets that are available on the Web. The checking and repair approach is generalizable in a way that it can be applied to any kind of dataset so that it can be consumed by any application that imposes a certain structure on the data.

We found that validation of datasets generated by (and for usage with) PPT, which are provided by either SWC or customers as well as datasets from the Web, can be done with reasonable performance. Furthermore, we believe that integrating repair strategies and data constraint specification helps in building a unified, maintainable model for expressing an application's requirements for reliably processing data from the Web. Such a model is also crucial for keeping compatibility between software versions and can play a pivotal role in harmonizing data and software development processes.

In our next steps, we will build on the foundations introduced in this paper. We will investigate methods for defining reusable repair strategies for constraint violations. The vision is that repair strategies, just as constraint definitions, may be published on the Web as Linked Data for anyone to adopt and extend. As constraint violation repair will, in some cases, require user input, we will also identify ways to generate user interfaces and best practices on how to repair a large number of constraint violations of the same kind.

Acknowledgements

This work was supported by grants from the EU's H2020 Programme ALIGNED (GA 644055).

References

1. Anicic, Darko et al. EP-SPARQL: A Unified Language for Event Processing and Stream Reasoning. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11. ACM, 2011.
2. H. Boley, A. Paschke, and O. Shafiq. Ruleml 1.0: The overarching specification of web rules. In *Semantic Web Rules - International Symposium, RuleML 2010, Washington, DC, USA, October 21-23, 2010. Proceedings*, pages 162–178, 2010.
3. A. Dimou, D. Kontokostas, M. Freudenberg, R. Verborgh, J. Lehmann, E. Mannens, S. Hellmann, and R. Van de Walle. *Assessing and Refining Mappingsto RDF to Improve Dataset Quality*, pages 133–149. Springer International Publishing, Cham, 2015.
4. C. Fürber and M. Hepp. Using sparql and spin for data quality management on the semantic web. In *Business Information Systems, 13th International Conference, BIS 2010, Berlin, Germany, May 3-5, 2010. Proceedings*, Lecture Notes in Business Information Processing. Springer, 2010.
5. J. B. Hansen, A. Beveridge, R. Farmer, L. Gehrmann, A. J. Gray, S. Khutan, T. Robertson, and J. Val. Validata: An online tool for testing rdf data conformance.
6. D. Kontokostas, P. Westphal, S. Auer, S. Hellmann, J. Lehmann, R. Cornelissen, and A. Zaveri. Test-driven evaluation of linked data quality. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 747–758. International World Wide Web Conferences Steering Committee, 2014.
7. C. Mader, B. Haslhofer, and A. Isaac. Finding quality issues in skos vocabularies. In *TPDL 2012 Therory and Practice of Digital Libraries*, Germany, May 2012.
8. A. Miles and S. Bechhofer. Skos simple knowledge organization system reference. WWW Consortium, Working Draft WD-skos-reference-20080829, August 2008.
9. A. Polleres. From sparql to rules (and back). In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07. ACM, 2007.
10. E. Prud'hommeaux, J. E. Labra Gayo, and H. Solbrig. Shape expressions: An rdf validation and transformation language. In *Proceedings of the 10th International Conference on Semantic Systems*, pages 32–40. ACM, 2014.
11. O. Suominen and E. Hyvönen. Improving the quality of skos vocabularies with skosify. In *Knowledge Engineering and Knowledge Management*, pages 383–397. Springer, 2012.
12. O. Suominen and C. Mader. Assessing and improving the quality of skos vocabularies. *Journal on Data Semantics*, 3(1):47–73, 2014.