Test-driven Evaluation of Linked Data Quality

Dimitris Kontokostas
University of Leipzig
kontokostas@informatik.uni-leipzig.de

Patrick Westphal
University of Leipzig
pwestphal@informatik.uni-leipzig.de

Sören Auer
University of Bonn and Fraunhofer IAIS
auer@cs.uni-bonn.de

Sebastian Hellmann
University of Leipzig
hellmann@informatik.uni-leipzig.de

Jens Lehmann
University of Leipzig
lehmann@informatik.uni-leipzig.de

Roland Cornelissen
Stichting Bibliotheek.nl
roland@metamatter.nl

ABSTRACT
Linked Open Data (LOD) comprises of an unprecedented volume of structured data on the Web. However, these datasets are of varying quality ranging from extensively curated datasets to crowd-sourced or extracted data of often relatively low quality. We present a methodology for test-driven quality assessment of Linked Data, which is inspired by test-driven software development. We argue, that vocabularies, ontologies and knowledge bases should be accompanied by a number of test-cases, which help to ensure a basic level of quality. We present a methodology for assessing the quality of linked data resources, based on a formalization of bad smells and data quality problems. Our formalization employs SPARQL query templates, which are instantiated into concrete quality test queries. Based on an extensive survey, we compile a comprehensive library of data quality test patterns. We perform automatic test instantiation based on schema constraints or semi-automatically enriched schemata and allow the user to generate specific test instantiations that are applicable to a schema or dataset. We provide an extensive evaluation of five LOD datasets, manual test instantiation for five schemas and automatic test instantiations for all available schemata registered with LOV. One of the main advantages of our approach is that domain specific semantics can be encoded in the data quality test cases, thus being able to discover data quality problems beyond conventional quality heuristics.

Categories and Subject Descriptors
H.2.0 [**DATABASE MANAGEMENT**]: General—*Security, integrity, and protection*; D.2.5 [**Software Engineering**]: Testing and Debugging —*Testing tools, Debugging aids*

Keywords
Data Quality, Linked Data, DBpedia

1. INTRODUCTION
Linked Open Data (LOD) comprises an unprecedented volume of structured data published on the Web. However, these datasets are of varying quality ranging from extensively curated datasets to crowd-sourced and even extracted data of relatively low quality. Data quality is not an absolute measure, but assesses fitness for use. Consequently, one of the main challenges regarding the wider deployment and use of semantic technologies on the Web is the assessment and ensuring of the quality of a certain possibly, evolving dataset for a particular use case. There have been few approaches for assessing Linked Data quality. However, these were majorly methodologies, which require (1) a large amount of manual configuration and interaction [2, 7, 16] or (2) automated, reasoning based methods [9, 11]. While reasoning based methods allow more automation, they are either limited to very specific quality aspects (such as link quality [9]) or lack scalability to the medium and large datasets being increasingly published as Linked Data. In consequence, we observe a shortage of practical quality assessment approaches for Linked Data, which balance between a high degree of automation *and* scalability to datasets comprising billions of triples.

In this article, we present a methodology for test-driven Linked Data quality assessment, which is inspired by test-driven software development. In software engineering, a *test-case* can be defined as *an input on which the program under test is executed during testing* and a *test-set* as *a set of test-cases for testing a program* [23]. A basic metric in software unit-testing is *test adequacy*, which measures the completeness of the test-set. A key principle of test-driven software development is to start the development with the implementation of automated test-methods before the actual functionality is implemented.

Compared to software source code testing, where test cases have to be implemented largely manually or with limited programmatic support, the situation for Linked Data quality testing is slightly more advantageous. On the Data Web we have a unified data model – RDF – which is the basis for both, data *and* ontologies. In this work we exploit the RDF data model by devising a pattern-based approach for the data quality tests of RDF knowledge bases. We argue, that ontologies, vocabularies and knowledge bases should be accompanied by a number of test-cases, which help to ensure a basic level of quality. We present a methodology for assessing the quality of linked data resources, based on a formalization of data quality integrity constraints. Our formalization employs SPARQL query templates, which are instantiated into concrete quality test queries. Based on an extensive survey, we compile a comprehensive library of quality test patterns, which can be instantiated for rapid development of more test cases. We provide a method for automatic test instantiation from these patterns for a particular ontology or vocabulary schema. Furthermore, we support the automatic derivation from OWL schema axioms. Since many schemata of LOD datasets are not very expressive, our methodology also includes semi-automatic schema enrichment. Concrete test cases are equipped with persistent identifiers to facilitate test tracking over time. We devise the notion of RDF test-case coverage based on a combination of six individual coverage metrics (four for properties and two for classes).

As a result, the test coverage can be explicitly stated for a certain dataset and potential users can thus obtain a more realistic assessment of the quality they can expect. Since the test-cases are related to certain parts of the knowledge base (i.e. properties and classes), also the quality of particular fragments relevant for a certain use-case can be easily assessed. Another benefit of test-driven data engineering is support for data evolution. Once test-cases are defined for a certain vocabulary, they can be applied to all datasets reusing elements of this vocabulary. Test-cases can be re-executed whenever the data is altered. Due to the modularity of the approach, where test-cases are bound to certain vocabulary elements, test-cases for newly emerging datasets, which reuse existing vocabularies can be easily derived.

Our approach allows to perform an automatic test instantiation based on schema constraints or semi-automatically enriched schemata and allows users to generate specific tests instantiations that are applicable for a schema or a dataset. A main contribution of our work is an extensive and unprecedented quantitative evaluation involving (a) manual and automatic test instantiations for five large-scale LOD datasets (two DBpedia editions, datos.bne.es, Library of Congress authority data and LinkedGeoData) and (b) automatic test instantiations for all available schemata registered with the *Linked Open Vocabularies* (LOV)[1] resulting in 32,293 total unique tests for 297 of the LOV vocabularies. One of the main advantages of our approach is that domain specific semantics can be encoded in the data quality test cases, thus being able to discover data quality problems beyond conventional quality heuristics. Finally, our framework implementation is built upon the SPARQL 1.1 standard which makes it applicable for any knowledge bases or triple store implementation.

The remainder of the article is structured as follows: Section 2 describes the methodology we followed to define *Data Quality Test Patterns*. The elicitation of our pattern library is described in Section 3. We instantiate, run and evaluate the tests in Section 4 and Section 5, followed by a discussion in Section 6. Section 7 elaborates on related work and we conclude in Section 8.

[1] `http://lov.okfn.org/`

2. TEST-DRIVEN DATA QUALITY METHODOLOGY

We first introduce the basic notions in our methodology, then describe its workflow and finally define test coverage criteria analogous to unit tests in software engineering.

Basic Notions

Data Quality Test Pattern (DQTP). A data quality test pattern is a tuple (V, S), where V is a set of typed pattern variables and S is a SPARQL query template with placeholders for the variables from V. Possible types of the pattern variables are IRIs, literals, operators, datatype values (e.g. integers) and regular expressions. With $R(v)$ we denote the value range for a pattern variable in $v \in V$, i.e. the set of values by which the variable can be substituted, and with $R(V)$ the union of all these sets, i.e. $R(V) = \bigcup_{v \in V} R(v)$.

Ideally, DQTPs should be knowledge base and vocabulary agnostic. Using `%%v%%` as syntax for placeholders, an example DQTP is:

```
SELECT ?s WHERE { ?s  %%P1%%   ?v1 .
                  ?s  %%P2%%   ?v2 .
                  FILTER ( ?v1 %%OP%% ?v2 ) }
```

This DQTP can be used for testing whether a value comparison of two properties $P1$ and $P2$ holds with respect to an operator OP. DQTPs represent abstract patterns, which can be further refined into concrete data quality test cases using test pattern bindings.

Test Pattern Binding. A test pattern binding is a specific instantiation of a DQTP. It is a triple (σ, S, C) in which $\sigma : V \to R(V)$ is a mapping of variables to valid replacements, S is a SPARQL query template and $C \in \{error, bad\_smell\}$ is used as classification of the error.

Data Quality Test Cases. Applying σ to S results in a SPARQL query, which can then be executed. Each result of the query is considered to be a violation of a unit test. An example test pattern binding and resulting data quality test case is[2]:

```
P1  =>  dbo:birthDate |   SELECT ?s WHERE {
P2  =>  dbo:deathDate |    ?s  dbo:birthDate  ?v1.
OP  =>  >             |    ?s  dbo:deathDate  ?v2.
                      |    FILTER ( ?v1 > ?v2 ) }
```

A test case has three different results: success (empty result), violation (results are returned) and timeout (test is marked for further inspection).

Test Auto Generators (TAG). Many knowledge bases use RDFS and OWL as modelling languages. While the core of those languages aims at inferring new facts, a number of constructs is also suitable for verifying data quality. In previous work, tools like the *Pellet Integrity Constraint Validator* [18] made use of this by viewing OWL axioms as constraints and reporting violations of them. Those are then

[2] We use `http://prefix.cc` to resolve all name spaces and prefixes. A full list can be found at `http://prefix.cc/popular/all`

interpreted via integrity constraint semantics, which uses a closed world assumption and a weaker form of the unique names assumption in which two individuals are considered to be different unless they are explicitly stated to be equal. We pursue the same approach for re-using schema information in our test framework. To achieve this, a test auto generator (TAG) takes a schema as input and returns test cases. We provide support for axioms such as `rdfs:domain`, `rdfs:range`, `owl:cardinality`, `owl:minCardinality`, `owl:maxCardinality`, `owl:functionalProperty`, `owl:disjointClass`, `owl:propertyDisjointWith`, `owl:complementOf`, `owl:InverseFunctionalProperty`, `owl:AsymmetricProperty` and `owl:IrreflexiveProperty`.

Generators consist of a detection and an execution part. The detection part is a query against a schema, for instance:

```
SELECT DISTINCT ?T1 ?T2 WHERE {
  ?T1 owl:disjointWith ?T2 . }
```

For every result of a detection query, a test-case is instantiated from the respective pattern, for instance:

```
SELECT DISTINCT ?s WHERE {
  ?s rdf:type %%T1%% .
  ?s rdf:type %%T2%% .}
```

Depending on the violation, there is not necessarily a one-to-one mapping between a detection query and the generated test cases. For the `owl:cardinality` constraint, for example, we use three TAGs: a TAG for the case a cardinality is 0, which checks whether the corresponding triple pattern is instantiated and two generators for values greater than 0, one to ensure that the property exists (TYPRODEP) and a second to validate the property occurrences (OWLCARD). Note that detection queries can be complex, although it should be noted that our goal is not to provide complete reasoning and constraint checking, but rather providing a lightweight mechanism verifying typical violations efficiently.

Workflow

Our methodology is illustrated in Figure 1. As shown in the figure, there are two major sources for creating tests. One source is stakeholder feedback from everyone involved in the usage of a dataset and the other source is the already existing RDFS/OWL schema of a dataset. Based on this, there are several ways in which tests can be created:

1. *Using RDFS/OWL constraints directly:* As previously explained, tests can be automatically created via TAGs in this case.
2. *Enriching the RDFS/OWL constraints:* Since many datasets provide only limited schema information, we perform automatic schema enrichment as recently researched in [4, 3]. Those schema enrichment methods can take an RDF/OWL dataset or a SPARQL endpoint as input and automatically suggest schema axioms with a certain confidence value by analysing the dataset. In our methodology, this is used to create further tests via TAGs. It should be noted that tests are explicitly labelled, such that the engineer knows that they are less reliable than manual tests.
3. *Re-using tests based on common vocabularies:* Naturally, a major goal in the Semantic Web is to re-use existing vocabularies instead of creating them from scratch for each dataset. We detect the used vocabularies in a dataset, which allows to re-use tests from a test pattern library. The creation of that library is described in the next section.
4. *Instantiate existing DQTPs:* The aim of DQTPs is to be generic, such that they can be applied to different datasets. While this requires a high initial effort of compiling a pattern library, it is beneficial in the long run, since they can be re-used. Instead of writing SPARQL templates themselves, an engineer can select and instantiate the correct DQTP. This does not necessarily require SPARQL knowledge, but can also be achieved via a textual description of a DQTP, examples and its intended usage.
5. *Write own DQTPs:* In some cases, test cases cannot be generated by any of the automatic and semi-automatic methods above and have to be written from scratch by an engineer. Those DQTPs can then become part of a central library to facilitate later re-use.

Test Coverage and Adequacy

In software engineering, a *test-case* can be defined as *an input on which the program under test is executed during testing* and a *test-set* as *a set of test-cases for testing a program* [23]. A basic metric in software unit-testing is *Test Adequacy.* According to [23], adequacy is a notion that measures the completeness of the test-set. An *Adequacy Stopping Rule* (ASR) is a related metric with a range $\{true|false\}$ that defines whether sufficient testing has been done. Many attempts have been made to quantify test adequacy with the main coverage criteria being: a) statement coverage, b) branch coverage, c) path coverage and d) mutation adequacy. It is hard to automate the creation of those tests.

In RDF, instead of code, the testing subject is *data* that is stored in triples and adheres to a schema. We define an RDF test-case as *a data constraint that involves one or more triples* and an RDF test-set as *a set of test cases for testing a dataset.* As there exist no branches and paths in RDF, a *test adequacy* metric can only be related to the selectivity of the test-cases. We will subsequently consider coverage as a composite of the following coverage criteria:

- *Property domain coverage (dom):* Identifies the ratio of property occurrences, where a test-case is defined for verifying domain restrictions of the property.
- *Property range coverage (ran):* Identifies the ratio of property occurrences, where a test-case is defined for verifying range restrictions of the property.
- *Property dependency coverage (pdep):* Identifies the ratio of property occurrences, where a test-case is defined for verifying dependencies with other properties.
- *Property cardinality coverage (card):* Identifies the ratio of property occurrences, where a test-case is defined for verifying the cardinality of the property.
- *Class instance coverage (mem):* Identifies the ratio of classes with test-cases regarding class membership.
- *Class dependency coverage (cdep):* Identifies the ratio of class occurrences for which test-cases verifying relationships with other classes are defined.

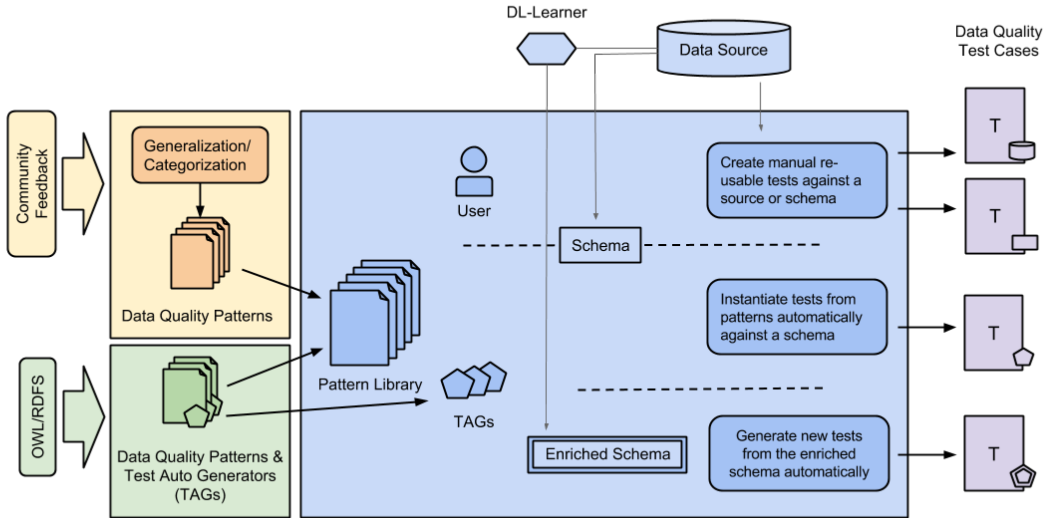

Figure 1: Flowchart showing the test-driven data quality methodology. The left part displays the input sources of our pattern library. In the middle part the different ways of pattern instantiation are shown which lead to the Data Quality Test Cases on the right.

A certain property should also be considered to be covered, if the absence of a particular constraint is explicitly stated.

The above criteria can be computed by *coverage computation functions*. Each coverage computation function $f : Q \rightarrow 2^E$ takes a SPARQL query $q \in Q$ corresponding to a test pattern binding as input and returns a set of entities. As an example, the function f_{dom} for computing the domain coverage returns the set of all properties p such that the triple pattern $(?s, p, ?o)$ occurs in q and there is at least one other triple pattern using $?s$ in q. This can straightforwardly be extended to a function $F : 2^Q \rightarrow 2^E$ taking a set of SPARQL queries as input and returning a set of entities. F computes how many entities are covered by the test queries. For properties, F can be further extended to a function F' with $F'(QS, D) = \sum_{p \in F(QS)} pfreq(p)$ where $pfreq(p)$ is the frequency of a property p, i.e. the number of occurrences of p divided by the number of occurrences of all properties in D. The extension for classes is analogous. This extension weights the entities by their frequency in the dataset. We propose to employ occurrences, i.e. concrete entity usages, instead of properties itself in order to reduce the influence of rarely used properties on the coverage.

The other coverage criteria are defined as follows: Range coverage f_{ran} is analogous to domain coverage. The property dependency coverage f_{pdep} of a query q returns all properties in q if there are at least two different properties and an empty set otherwise. Property cardinality coverage f_{card} of a query q returns the set of all properties p, such that $(?s, p, ?o)$ occurs in q along with `GROUP BY ?s` as well as `HAVING(count(?s) op n)` aggregates (op is one of $\leq, <, =, >, \geq$ and n a number) or, analogously, the same criteria for $?o$ instead of $?s$. Class instance coverage f_{mem} of a query q returns the set of all classes c such that $(?s, \mathit{rdf{:}type}, c)$ occurs in q. The class dependency coverage f_{cdep} of a query q returns all classes in q if there are at least two different classes and an empty set otherwise.

In the above definition, please note that domain and range restrictions are more general then verifying `rdfs:domain` and `rdfs:range` as they cover all tests, which can be performed via SPARQL on subject and objects of triples using a particular property. Please note that many tests can be expressed in OWL 2, in particular when using the Pellet integrity constraint semantics. For instance, custom datatypes in OWL2[3] can be used for range checking of property values using regular expressions. As noted above, we transparently support the usage of OWL, but some tests are much easier to implement in SPARQL and others, e.g. the SKOS restriction "A resource has no more than one value of skos:prefLabel per language tag." cannot be checked in OWL at all, but are a straightforward DQTP in our case (ONELANG in Table 1).

Formally, we can define RDF test-case coverage Cov of a set of test queries QS with respect to a dataset D as follows:

$$\begin{aligned} Cov(QS, D) = &\frac{1}{6}(F'_{dom}(QS, D) + F'_{ran}(QS, D) \\ &+ F'_{pdep}(QS, D) + F'_{card}(QS, D) \\ &+ F'_{mem}(QS, D) + F'_{cdep}(QS, D)) \end{aligned}$$

The coverage is a heuristic, which helps to assess whether the defined test cases are sufficient for data quality assessment.

3. PATTERN ELICITATION AND CREATION

To start capturing patterns of real data errors we had a closer look at the DBpedia being one of the bigger and best interlinked datasets in the LOD cloud [15]. We performed three different analyses which led to a comprehensive library of test patterns summarized in Table 1:

1. Analysis of incidental error reports by the DBpedia user community.
2. Analysis of error tracking behavior by Wikipedia editors.

[3] http://www.w3.org/TR/owl2-primer/#Advanced_Use_of_Datatypes

3. Analysis of the ontology schema of the DBpedia OWL ontology.

Community feedback. We thoroughly reviewed all the DBpedia related mailing lists and QA websites, i.e. the *DBpedia discussion*[4] and *DBpedia developers*[5] lists, as well as questions tagged with *DBpedia* on *stackoverflow*[6] and *Semantic Web Answers*[7]. We picked all the data quality related questions and tried to create SPARQL queries for retrieving the same erroneous data. Finally, we grouped similar SPARQL queries together.

Wikipedia maintenance system. We reviewed the information Wikipedia uses to ensure article quality and tried to reuse it from DBpedia. Such information encompasses special *Categories* and *Templates* used by seasoned Wikipedians (e.g. admins and stewards) to administrate and tag errors in the article space[8]. Based on the maintenance categories and templates used, new patterns like the TRIPLE Pattern and the PVT Pattern were derived. These patterns are also applicable to other datasets, e.g. LinkedGeoData [19].

OWL ontology analysis. The main purpose of OWL is to infer knowledge from existing schemata and data. While it can also be used to check constraints, this can be difficult in practice due to the Open World Assumption used and the lack of the Unique Name Assumption. Therefore, in addition to standard OWL inference, it can also be useful to convert OWL ontology axioms to SPARQL queries, which check the constraints expressed by them. This is motivated by research on the *Pellet Integrity Constraint Validator*[9] using the same idea. Specifically, we analysed the ontology and checked which existing constructs are applicable for constraint checking in DBpedia. We identified constructs such as (inverse) functionality, cardinality, domain, and range of properties as well as class disjointness as relevant and included them in our pattern template library. The bindings for those patterns can be created automatically from specific OWL ontology axioms.

Pattern Library

Our *Pattern Library* consists of 17 DQTPs. Table 1 shows a description of all patterns along with two example bindings. In the following, we exemplarily illustrate two patterns in detail and refer the reader to `http://svn.aksw.org/papers/2014/WWW_Databugger/public.pdf` for a complete pattern description.

COMP Pattern . Depending on the property semantics, there are cases where two different literal values must have a specific ordering with respect to an operator. *P1* and *P2* are the datatype properties we need to compare and *OP* is the comparison operator $\mathrm{R}(OP) =$ `{ <, <=, >, >=, =, != }`.

```
SELECT ?s WHERE { ?s  %%P1%%  ?v1 .
                  ?s  %%P2%%  ?v2 .
                  FILTER ( ?v1 %%OP%% ?v2 ) }
```

[4] https://lists.sourceforge.net/lists/listinfo/dbpedia-discussion

[5] https://lists.sourceforge.net/lists/listinfo/dbpedia-developers

[6] http://stackoverflow.com/questions/tagged/dbpedia

[7] http://answers.semanticweb.com/tags/dbpedia/

[8] http://en.wikipedia.org/wiki/Category:Wikipedia_maintenance

[9] http://clarkparsia.com/pellet/icv/

Schema	Tests	Schema	Tests
dicom	8,229	mo	605
dbo	5,713	tio	525
frbrer	2,166	uco	516
biopax	688	vvo	506
hdo	682	ceo	511

Table 2: Top 10 schemas with descending number of automatically generated tests.

Example bindings: (a) `dbo:deathDate` before '<' `dbo:birthDate`, (b) `dbo:releaseDate` after '>' `dbo:latestReleaseDate`.

MATCH Pattern. Application logic or real world constraints may put restrictions on the form of a literal value. *P1* is the property we need to check against *REGEX* and *NOP* can be a not operator ('!') or empty.

```
SELECT ?s WHERE { ?s  %%P1%%  ?value .
   FILTER ( %%NOP%% regex(str(?value), %%REGEX%) ) }
```

Example bindings: (a) `dbo:isbn` format is different '!' from "ˆ([iIsSbBnN 0-9-])*$" (b) `dbo:postCode` format is different '!' from "ˆ[0-9]{5}$".

4. TEST GENERATION

To evaluate our methodology, we automatically generated test cases for all available vocabularies in the LOV dataset. Using the implemented TAGs, we managed to create 32,293 total unique reusable test cases for 297 LOV vocabularies[10]. Table 2 displays the 10 schemas with the most associated tests. For brevity we use the vocabulary prefixes as defined in LOV.[11] Tests are themselves described in RDF and have a stable URI for tracking them over time. The URI is generated under the application namespace concatenated with the schema prefix, the pattern and an MD5 checksum of the SPARQL query string. The following listing displays a test that checks whether the domain of `foaf:primaryTopic` instance is a `foaf:Document`. We store metadata along with every test case which allows us to easily filter test cases based on different criteria.

```
tddt:foaf-RDFSDOMAIN-8e121cf1111201b5d53de161e245c137
 a tddo:Test ;
 tddo:appliesTo tddo:Schema ;
 tddo:basedOnPattern tddp:RDFSDOMAIN ;
 tddo:generated tddo:AutoGenerated ;
 tddo:references foaf:Document , foaf:primaryTopic ;
 tddo:source <http://xmlns.com/foaf/0.1/> ;
 tddo:sparql """ SELECT DISTINCT ?s WHERE {
  ?s foaf:primaryTopic> ?v1
  FILTER NOT EXISTS {?s rdf:type+ foaf:Document}}""";
 tddo:sparqlPrevalence
  """SELECT (count( DISTINCT ?s) AS ?total) WHERE {
  ?s foaf:primaryTopic ?c .} """ ;
 tddo:testGenerator tddg:RDFSDOMAIN .
```

[10] LOV had 367 vocabularies at the date of last access (5/10/2013) but not all were accessible.

[11] In addition to the LOV schemas, dbo (`http://dbpedia.org/ontology/`), frbrer (`http://iflastandards.info/ns/fr/frbr/frbrer/`) and isbd (`http://iflastandards.info/ns/isbd/elements/`) schemas are included as prefixes.

Pattern	Description	Type	Binding example
COMP	Comparison between two literal values of a resource.	dom ran pdep	a) `dbo:deathDate` before `dbo:birthDate`
			b) `dbo:releaseDate` after `dbo:latestReleaseDate`
MATCH	The literal value of a resource matches/does not match a certain regex pattern	ran	a) `dbo:isbn` does not match "^[0-9]{5}$"
			b) `foaf:phone` contains any letters ("[A-Za-z]")
LITRAN	The literal value of a resource (having a certain type) must (not) be within a specific range	dom ran pdep mem	a) `dbo:height` of a `dbo:Person` is not within [0.4,2.5]
			b) `geo:lat` of a `spatial:Feature` is not within [-90,90]
TYPEDEP	Type dependency: The type of a resource may imply the attribution of another type.	dom cdep	a) a resource is a `gml:_Feature` but not a `dbo:Place`
			b) a resource is a `foaf:Person` but not a `dbo:Person`
TYPRO-DEP	A resource of a specific type should have a certain property.	dom pdep mem	a) a `foaf:Document` should have a `foaf:primaryTopic`
			b) a `dbo:Person` should have a `dbo:birthDate`
PVT	If a resource has a certain value V assigned via a property P_1 that in some way classifies this resource, one can assume the existence of another property P_2.	dom pdep	a) DBpedia articles stemming from a *Geographic_location* template should have coordinates assigned via `georss:point`
			b) DBpedia resources in the category *1907 births* should have a `dbo:birthDate`
TRIPLE	A resource can be considered erroneous if there are corresponding hints contained in the dataset		a) resources stemming from Wikipedia articles that are possibly copy-pasted (i.e. having the category *Possible_cut-and-paste_moves* assigned)
			b) geographical features (of the linkedgeodata.org dataset) that are marked with the `lgdo:fixme` property
ONELANG	A literal value should contain at most one literal for a certain language.	ran card	a) a resource should only have one English `foaf:name`
			b) a resource should only have one English `rdfs:label`
RDFS-DOMAIN	The attribution of a resource's property (with a certain value) is only valid if the resource is of a certain type.	dom pdep mem	a) a resource having a `dbo:demographicsAsOf` property not being a `dbo:PopulatedPlace`
			b) a resource has the "Cities of Africa" category assigned but is not of type `dbo:City`
RDFS-RANGE	The attribution of a resource's property is only valid if the value is of a certain type	ran pdep mem	a) a `dbo:Person`'s spouse not being a `dbo:Person`
			b) a resource assigned via the `foaf:based_near` property not being of type `geo:SpatialThing`
RDFS-RANGED	The attribution of a resource's property is only valid if the literal value has a certain datatype	ran pdep mem	a) the value of the property `dbo:isPeerReviewed` must be of type `xsd:boolean`
			b) the value of the property `dbo:successfulLaunches` must be of type `xsd:nonNegativeInteger`
INVFUNC	Some values assigned to a resource are considered to be unique for this particular resource and must not occur in connection with other resources.	ran	a) there must not be more than one resource with the same `foaf:homepage`
			b) there must not be more than one country with the same `dbo:capital`
OWL-CARD	Cardinality restriction on a property	ran card	a) `dbo:birthDate` is a functional property
			b) there should be just one `skos:prefLabel`
OWLDISJC	Disjoint class constraint	cdep	a) a `foaf:Document` is disjoint with `foaf:Project`
			b) a `dbo:Person` is disjoint with `dbo:Work`
OWLDISJP	Disjoint property constraint	dom ran pdep mem	a) `skos:prefLabel` is disjoint with `skos:hiddenLabel`
			b) `dbo:bandMember` is disjoint with `dbo:birthPlace`
OWL-ASYMP	Asymmetric property constraint	dom ran	a) `dbo:child` is asymmetric
			b) `dbo:birthPlace` is asymmetric
OWL-IRREFL	Irreflexive property constraint	dom ran	a) `dbo:parent` is irreflexive
			b) `dbo:child` is irreflexive

Table 1: Example templates and bindings. The column *Type* refers to the coverage type.

Schema	Tests	Schema	Tests
dbpedia.org	1,723	id.loc.gov	48
nl.dbpedia.org	845	datos.bne.org	18
linkedgeodata.org	61		

Table 3: Number of additional tests instantiated for the enriched schemas.

Pattern	Tests	Manual
RDFSDOMAIN	16,645	3
RDFSRANGE	9.727	4
OWLDISJC	5,530	-
EDFSRANGED	5,073	-
OWLDISJP	1,813	10
OWLCARD	1,818	6
TYPRODEP	746	13
OWLASYMP	660	-
OWLIRREFL	342	-
INVFUNC	338	1
MATCH	9	9
LITRAN	5	5
COMP	4	4
ONELANG	4	4
PROPDEP	4	4
TYPDEP	2	2
TRIPLE	2	2

Table 4: Number of total and manual tests per pattern for all LOV vocabularies.

For every dataset evaluated, we applied automatic schema enrichment as described in Section 2. We used a high level of confidence (0.9; see [4] for details) on the produced axioms and applied manual post-processing to remove certain axioms. The number of additional tests instantiated for the considered schemas are shown in Table 3.

Besides the automatically generated test cases, our methodology supports manual tests that may apply to a schema or a dataset. The manual schema tests are reusable across different datasets for all RDF data using that schema. The manual dataset tests can be applied only to a specific dataset. Manual tests usually require domain knowledge, which the authors have for a subset of the evaluation datasets. For the purposes of this evaluation, we defined 22 manual tests for the DBpedia ontology (`dbo`), six for the LinkedGeoData ontology (`lgdo`), three for the WGS84 Geo Positioning ontology (`geo`) as well as 15 manual tests for the DBpedia in English dataset. Additionally, we defined 20 manual tests for the `SKOS` vocabulary exploiting existing domain expertise [20]. Table 4 presents an aggregation of the defined tests based on the pattern they stem from.

5. LINKED DATA QUALITY EVALUATION

To showcase the re-usability of our automatically and manually generated test cases, we chose the following datasets for evaluation:

- **`dbpedia.org`**[12] extracts data from the English Wikipedia and publishes the data using the following schemas: **`owl`**, **`dbo`**, **`foaf`**, **`dcterms`**, **`dc`**, **`skos`**, **`geo`** and **`prov`** [15].
- **`nl.dbpedia.org`**[13] extracts data from the Dutch Wikipedia edition using the same vocabularies as the English DBpedia.
- **`linkedgeodata.org`**[14] provides a linked data mirror of OpenStreetMap[15] using the following schemas: **`ngeo`**, **`spatial`**, **`lgdo`**, **`dcterms`**, **`gsp`**, **`owl`**, **`geo`**, **`skos`** and **`foaf`** [19].
- **`id.loc.gov`**[16], is a SKOS dataset that publishes Library of Congress authority data using **`owl`**, **`foaf`**, **`dcterms`**, **`skos`**, **`mads`**, **`mrel`** and **`premis`** schemas.
- **`datos.bne.es`**[17], provides open bibliographic linked data from the Spanish National Library using **`owl`**, **`frbrer`**, **`isbd`**, **`dcterms`** and **`skos`** schemas.

To identify the schemas for each dataset, we used existing information from the *LODStats* project[18] [6]. The English (`dben`) and Dutch (`dbnl`) DBpedia share a similar structure, but the actual data differs [13]. Both DBpedia and the LinkedGeoData (`lgd`) datasets are generated from crowd-sourced content and thus are prone to errors. The Library of Congress authority data (`loc`) and the Open bibliographic data from the Spanish National Library (`datos`) were chosen as high quality bibliographic datasets with `loc` focusing on publishing SKOS and in the case of `datos` FRBR[19] data. The DBpedia datasets were tested using their online SPARQL endpoints and the other three datasets were loaded in a local triple store.[20]

Table 5 provides an overview of the dataset quality evaluation. In Table 6 we present the total errors aggregated per schema and in Table 7 the total errors aggregated per pattern. The test coverage for every dataset is provided in Table 8.

The occurrence of a high number of errors in the English DBpedia is attributed to the data loaded from external sources. For example, the recent load of transformed Wikidata data[21] almost doubled `rdfs:domain` and `rdfs:range` violations and errors in the `geo` schema. A common error in DBpedia is the `rdfs:range` violation. Triples are extracted from data streams and complete object range validation cannot occur at the time of extraction. Example violations from the `dbo` schema are the `rdfs:domain` of `dbo:sex` (1M) and `dbo:years` (550K) properties. Other `dben` errors based on the `foaf` schema are attributed mainly to the incorrect `rdfs:domain` or `rdfs:range` of `foaf:primaryTopic` (12M), `foaf:isPrimaryTopicOf` (12M) `foaf:thumbnail` (3M) and `foaf:homepage` (0.5M).

Among errors from the manual tests created for the DBpedia ontology are the following:

- 163K (102K in `dbnl`) resources with wrong postal code format.

[12] `http://dbpedia.org` (version 3.9)

[13] `http://nl.dbpedia.org` (live version, accessed on 05/10)

[14] `http://downloads.linkedgeodata.org/releases/2013-08-14/`

[15] `http://www.openstreetmap.org`

[16] `http://id.loc.gov/download/` (accessed on 05/10/2013)

[17] `http://datos.bne.es/datadumps/`, (accessed on 05/10/2013)

[18] `http://stats.lod2.eu/`

[19] `www.oclc.org/research/activities/frbr.html`

[20] We used the Virtuoso V7 triple store, because it supports SPARQL 1.1 property paths.

[21] `http://www.mail-archive.com/dbpedia-discussion@lists.sourceforge.net/msg05583.html`

Dataset	Triples	Subjects	Tests	Pass	Fail	TO	Errors	ManEr	EnrEr	E/R
dbpedia.org	817,467,330	24,922,670	6,064	4,288	1.860	55	63,644,169	5,224,298	249,857	2.55
nl.dbpedia.org	74,790,253	4,831,594	5,173	4,149	812	73	5,375,671	211,604	15,041	1.11
linkedgeodata.org	274,690,851	51,918,417	634	545	86	3	57,693,912	133,140	1	1.11
datos.bne.es	60,017,091	7,470,044	2,473	2,376	89	8	27,943,993	25	537	3.74
id.loc.gov	436,126,273	53,072,042	536	499	28	9	9,392,909	49	3,663	0.18

Table 5: Evaluation overview for the five tested datasets. For every dataset we display the total number of triples and the distinct number of subjects. We mention the total number of tests that run on each dataset, how many tests passed, failed and did timeout (TO). Finally we show the total number of errors, as well the total number of errors that occurred from manual (ManEr) and enriched (EnrEr) tests. The last column shows the average errors per distinct subject.

- 7K (137 in `dbnl`) books with wrong ISBN format.
- 40K (1.2K in `dbnl`) persons with a death date and without birth date.
- 638K persons without a birth date.
- 197K places without coordinates.
- 242K resources with coordinates that are not a `dbo:Place`.
- 28K resources with exactly the same coordinates with another resource.
- 9 resources with invalid longitude.

The `lgd` dataset also has a high number of errors per resource. Although the LinkedGeoData ontology is big, the information of interest for our methodology is mostly limited to `rdfs:domain` and `rdfs:range` axioms. Because of its broad vocabulary, mostly stemming from curated crowd-sourced user input, only a few manual test cases were found. In-depth domain knowledge is required to define further tests. These resulted in 132K errors for resources with a `lgdo:fixme` predicate and 250 with `lgdo:todo`, 637 wrong phone numbers and 22 resources having a `lgdo:start` property but no `lgdo:end`.

The `datos` dataset yielded a total of 28 million errors. In absolute numbers, `rdfs:domain` and `rdfs:range` violations were dominant. The `isbd:P1016` and `isbd:P1185` properties produced the most `rdfs:domain` violations (2.38M and 2.35M respectively). The schemas used in `datos` are expressive and there were many violations stemming from `owl:disjointWith` and `owl:propertyDisjointWith` constraints. With regards to the manual errors, 6 occurred due to shared literals between `skos:prefLabel` and `skos:altLabel` [20] and 25 because of property disjointness violations between `skos:broader`, `skos:narrower` and `skos:related`.

The `loc` dataset generated a total of 9 million errors. However, 99.9% originated from one test case: the `rdfs:domain` of `skos:member`. Other minor errors occurred in other schemas (cf. Table 6), e.g. incorrect `rdfs:domain` of `skos:topConceptOf` and incorrect `rdfs:domain` of `foaf:focus`. Similar to the `datos` dataset, 49 manual errors occurred from disjoint properties between `skos:broader`, `skos:narrower` and `skos:related`.

The highest *test coverage* is found in the `datos` dataset. This is due to the rich `frbrer` and `isbd` schemas. Although `dben` had a bigger test set than `datos`, it publishes a lot of automatically generated properties under the `dbp` namespace [15, Section 2] which lowers the coverage scores. The low

		Errors				
Schema	**Tests**	**dben**	**dbnl**	**lgd**	**datos**	**loc**
`dbo`	5,712	7.9M	716K	-	-	-
`frbrer`	2,166	-	-	-	11K	-
`lgdo`	224	-	-	2.8M	-	-
`isbd`	179	-	-	-	28M	-
`prov`	125	25M	-	-	-	-
`foaf`	95	25M	4.6M	-	-	59
`gsp`	83	-	-	39M	-	-
`mads`	75	-	-	-	-	280K
`owl`	48	5	3	2	5	-
`skos`	28	41	-	-	-	9M
`dcterms`	28	960	881	191K	37K	659
`ngeo`	18		-	119	-	-
`geo`	7	2.8M	120K	16M	-	-

Table 6: Total errors in the evaluated datasets per schema.

test coverage for `lgd` can be attributed to the very large but relatively flat and inexpressive schema. For DBpedia in Dutch we evaluated the Live endpoint and thus could not calculate property and class occurrences.

6. DISCUSSION

The most frequent errors in all datasets were produced from `rdfs:domain` and `rdfs:range` test cases. Domain and range are two of the most commonly expressed axioms in most schemas and, thus, produce many automated test cases and good test coverage. Errors from such violations alone cannot classify a dataset as low quality. In DBpedia, a resource is generated for every Wikipedia page and connected with the original article through the `foaf:primaryTopic`, `foaf:isPrimaryTopicOf` and `prov:wasDerivedFrom` predicates. DBpedia neither states that the Wikipedia page is a `foaf:Document` nor that the DBpedia resource a `prov:Entity`, as the FOAF and PROV vocabularies demand. This produced a total of 33 million errors (35% of the total errors) in the English DBpedia. In most cases, fixing such errors is easy and dramatically reduces the error rate of a dataset. However, DBpedia, as well as most LOD datasets, do not load all the schemas they reference in their endpoints. Thus, locating such errors by using only local knowledge is not effective, whereas our pattern library can be used without further overhead.

Testing for external vocabularies. According to our methodology, a dataset is tested against all the

Pattern	dben	dbnl	lgd	datos	loc
COMP	1.7M	7	-	-	-
INVFUNC	279K	13K	-	511	3.5K
LITRAN	9	-	-	-	-
MATCH	171K	103K	637	-	-
OWLASYMP	19K	3K	-	-	-
OWLCARD	610	291	1	1	3
OWLDISJC	92	-	-	8.1K	1.1K
OWLDISJP	3.4K	7K	-	53	223
OWLIRREFL	1.4K	14	-	-	-
PVT	267K	1.2K	22	-	-
RDFSDOMAIN	31M	2.3M	55M	28M	9M
RDFSRANGE	26M	2.5M	191K	320K	111K
RDFSRANGED	760K	286K	2.7M	2	-
TRIPLE	-	-	132K	-	-
TYPDEP	674K	-	-	-	-
TYPRODEP	2M	100K	-	-	-

Table 7: Total errors per pattern.

Metric	dben	lgd	datos	loc
f_{pdom}	20.32%	8.98%	72.26%	20.35%
f_{pran}	23.67%	10.78%	37.64%	28.78%
f_{pdep}	24.93%	13.65%	77.75%	29.78%
f_{card}	23.67%	10.78%	37.63%	28.78%
f_{mem}	73.51%	12.78%	93.57%	58.62%
f_{cdep}	37.55%	0%	93.56%	36.86%
$Cov(QS, D)$	33.94%	9.49%	68.74%	33.86%

Table 8: Test coverage on the evaluated datasets.

schemas it references. Although this approach provides better testing coverage, it can be insufficient when testing against unused data. Properties like `foaf:weblog` that do not exist in neither evaluated dataset, auto-generate 3 tests for `rdfs:domain`, `rdfs:range` and `owl:InverseFunctionalProperty`. In the future, the methodology could be refined to intelligently pre-process a dataset and reduce the number of test cases to run.

Revision of manually instantiated patterns. Although our pattern library already covers a wide range of data quality errors, there are cases where the mere instantiation of patterns is not sufficient. Binding COMP-a (cf. Table 1), for example, returns 509 results in the English DBpedia. Some of these results have, however, incomplete dates (i.e. just `xsd:gMonthDay`). Technically, these results are outside of the scope of the binding and the pattern and, therefore, a false positive. This can only be resolved by writing manual tests or adding another DQTP. In this scenario, the extended test could be as follows:

```
SELECT COUNT(*) WHERE { ?s dbo:birthDate ?v1 .
                        ?s dbo:deathDate ?v2 .
  FILTER (?v1>?v2 && datatype(?v1)!=xsd:gMonthDay
                  && datatype(?v2)!=xsd:gMonthDay) }
```

While axioms in an OWL ontology are intended to be applicable in a global context, our test-driven methodology also depends on domain knowledge to capture more semantics in data. However, there are cases where data constraints can be very application specific and not universally valid. For instance, due to the vast size of DBpedia, it is unrealistic to expect completeness, e.g. that every `dbo:Person` has a `foaf:depiction` and a `dbo:birthDate`. However, in the context of an application like “A day like today in history”[22] these properties are mandatory. Thus, a refinement of the methodology could support manual tests cases associated for an application context.

The software used to generate the tests and produce the evaluation results is available as open source[23]. At the project website[24], we provide a user interface and a dump of all results as RDF .

7. RELATED WORK

Previous Data Quality Measurements on DBpedia. The first publication of DBpedia [1] mainly concentrates on the data source – Wikipedia. Errors in the RDF data are attributed to several shortcomings in the authoring process, e.g. the usage of tables instead of templates, the encoding of layout information like color in templates and so on. Other inaccuracies occur due to an imprecise use of the wiki markup or when duplicate information is given, as in *height = 5’11” (180cm)*. To avoid those errors the authors provide some authoring guidelines in accordance with guidelines created by the Wikipedia community.
In [14], the authors concentrate more on the extraction process, comparing the *Generic* with the *Mapping-based Infobox Extraction* approach. It is shown that by mapping Wikipedia templates to a manually created, simple ontology, one can obtain a far better data quality, eliminating data type errors as well as a better linkage between entities of the dataset. Other errors concern class hierarchies e.g. omissions in the automatically created YAGO classification schema.
Another issue already addressed in the future work section of [14] is the fusion of cross-language knowledge of the language specific DBpedia instances. This topic as well as other internationalization issues are treated in [13]. There, different extraction problems of the Greek DBpedia are presented that can also be applied to other languages, especially those using non-Latin characters.
Another study aimed to develop a framework for the DBpedia quality assessment is presented in [21] and involves a manual and a semi-automatic process. In the manual phase the authors detects common problems and classify them in a taxonomy. After that, they crowdsource the evaluation of a large number of individual resources and let users structure it according to their taxonomy.

General Linked Data Quality Assessment. There exist several approaches for assessing the quality of Linked Data. We only give a brief overview here and refer to [22] for details. Approaches can be broadly classified into (i) automated (e.g. [9]), (ii) semi-automated (e.g. [7]) or (iii) manual (e.g. [2, 16]) methodologies. These approaches are useful at the process level wherein they introduce systematic methodologies to assess the quality of a dataset. However, the drawbacks include a considerable amount of user

[22] `http://el.dbpedia.org/apps/DayLikeToday/`
[23] `http://github.com/AKSW/Databugger`
[24] `http://databugger.aksw.org`

involvement, inability to produce interpretable results, or not allowing a user the freedom to choose the input dataset. In our case, we focused on a very lightweight framework and the development of a library based on real user input.
Additionally, there have been efforts to assess the quality of Web Data [5] on the whole, which included the analysis of 14.1 billion HTML tables from Google's general-purpose web crawl in order to retrieve tables with high-quality relations. In a similar vein, in [10], the quality of RDF data was assessed. This study detected the errors occurring while publishing RDF data along with the effects and means to improve the quality of structured data on the web. In a recent study, 4 million RDF/XML documents were analyzed which provided insights into the level of conformance these documents had in accordance to the Linked Data guidelines. On the one hand, these efforts contributed towards assessing a vast amount of Web or RDF/XML data, however, most of the analysis was performed automatically, therefore overlooking the problems arising due to contextual discrepancies. In previous work, we used similar ideas for describing the evolution of knowledge bases [17].

Rules and SPARQL. The approach described in [8] advocates the use of SPARQL and SPIN for RDF data quality assessment and shares some similarity with our methodology. However, a domain expert is required for the instantiation of test patterns. *SPARQL Inferencing Notation* (SPIN) [12] is a W3C submission aiming at representing rules and constraints on Semantic Web models. SPIN also allows users to define SPARQL functions and reuse SPARQL queries. The difference between SPIN and our pattern syntax, is that SPIN functions would not fully support our *Pattern Bindings*. SPIN function arguments must have specific constraints on the argument datatype or argument class and do not support operators, e.g. '=', '>', '!', '+', '*', or property paths[25]. However, our approach is still compatible with SPIN when allowing to initialise templates with specific sets of applicable operators. In that case, however, the number of templates increases. Due to this restrictions, SPIN defines fewer but more general constraints. The following SPIN example[26] tries to locate all the `owl:disjointWith` constraint violations.

```
SELECT ?x WHERE { ?c1 owl:disjointWith ?c2 .
                  ?x a ?c1 .
                  ?x a ?c2 . }
```

The problems of this type of query is that: 1) they are more expensive to execute, 2) aggregate all errors in a single result which makes it harder to debug and 3) cannot capture violations like `foaf:primaryTopic` if the `foaf` schema is not loaded in the knowledge base itself.
One of the advantages of converting our templates to SPIN is that the structure of the SPARQL query itself can be stored directly in RDF, which, however, renders it more complex. From the efforts related to SPIN, we re-used their existing data quality patterns and ontologies for error types.
Another related approach is the *Pellet Integrity Constraint Validator* (ICV)[27]. *Pellet ICV* [18] translates OWL integrity constraints into SPARQL queries. Similar to our approach, the execution of those SPARQL queries indicates violations. An implication of the integrity constraint semantics of Pellet ICV is that a partial unique names assumption (all resources are considered to be different unless equality is explicitly stated) and a closed world assumption is in effect. We use the same strategy as part of our methodology, but go beyond it by allowing users to directly (re-)use DQTPs not necessarily encoded in OWL and by providing automatic schema enrichment.
Schemarama[28] is a very early (2001) constraint validation approach based on using the Squish RDF language instead of SPARQL. It does not offer a templating mechanism or a classification of data quality problems.
For XML, *Schematron*[29] is an ISO standard for validation and quality control of XML documents based on XPath and XSLT. We argue that similar adapted mechanisms for RDF are of crucial importance to provide solutions allowing the usage of RDF in settings, which require either high quality data or at least an accurate assessment of its quality.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we described a novel approach for improving Linked Data quality. The approach is inspired by test-driven software engineering and is centered around the definition of data quality integrity constraints, which are represented in SPARQL query templates. We compiled a comprehensive set of generic Data Quality Test Patterns (DQTP), which we instantiated for 297 schemas resulting in 32,293 test cases. We reused these test cases to evaluate the quality of five LOD datasets. Our evaluation showed that DQTPs are able to reveal a substantial amount of data quality issues in an effective and efficient way.

We see this work as the first step in a larger research and development agenda to position test-driven data engineering similar to test-driven software engineering. In future work, we aim to tackle automatic repair strategies, i.e. how can templates and bindings be used to fix problems efficiently. We also plan to implement a test-driven data quality cockpit, which allows users to easily instantiate and run DQTPs based on custom knowledge bases. As a result, we hope that test-driven data quality can contribute to solve one of the most pressing problems of the Data Web – the improvement of data quality and the increase of Linked Data fitness for use.

Acknowledgment

This work was supported by grants from the European Union's 7th Framework Programme provided for the projects LOD2 (GA no. 257943), GeoKnow (GA no. 318159) and DIACHRON (GA no 601043).

[25] `http://www.w3.org/TR/2010/WD-sparql11-property-paths-20100126/`
[26] `http://topbraid.org/spin/owlrl-all.html#cax-dw`
[27] `http://clarkparsia.com/pellet/icv/`
[28] `http://swordfish.rdfweb.org/discovery/2001/01/schemarama/`
[29] `http://www.schematron.com/`

9. REFERENCES

[1] S. Auer and J. Lehmann. What have Innsbruck and Leipzig in common? extracting semantics from wiki content. In *Proceedings of the ESWC (2007)*, volume 4519 of *Lecture Notes in Computer Science*, pages 503–517, Berlin / Heidelberg, 2007. Springer.

[2] C. Bizer and R. Cyganiak. Quality-driven information filtering using the WIQA policy framework. *Web Semantics*, 7(1):1 – 10, Jan 2009.

[3] L. Bühmann and J. Lehmann. Universal OWL axiom enrichment for large knowledge bases. In *Proceedings of EKAW 2012*, pages 57–71. Springer, 2012.

[4] L. Bühmann and J. Lehmann. Pattern based knowledge base enrichment. In *12th International Semantic Web Conference, 21-25 October 2013, Sydney, Australia*, 2013.

[5] M. J. Cafarella, A. Y. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: exploring the power of tables on the web. *PVLDB*, 1(1):538–549, 2008.

[6] J. Demter, S. Auer, M. Martin, and J. Lehmann. Lodstats – an extensible framework for high-performance dataset analytics. In *Proceedings of the EKAW 2012*, Lecture Notes in Computer Science (LNCS) 7603. Springer, 2012. 29

[7] A. Flemming. Quality characteristics of linked data publishing datasources. Master's thesis, Humboldt-Universität of Berlin, 2010.

[8] C. Fürber and M. Hepp. Using sparql and spin for data quality management on the semantic web. In W. Abramowicz and R. Tolksdorf, editors, *BIS*, volume 47 of *Lecture Notes in Business Information Processing*, pages 35–46. Springer, 2010.

[9] C. Guéret, P. T. Groth, C. Stadler, and J. Lehmann. Assessing linked data mappings using network measures. In *Proceedings of the 9th Extended Semantic Web Conference*, volume 7295 of *Lecture Notes in Computer Science*, pages 87–102. Springer, 2012.

[10] A. Hogan, A. Harth, A. Passant, S. Decker, and A. Polleres. Weaving the pedantic web. In *LDOW*, 2010.

[11] Q. Ji, P. Haase, G. Qi, P. Hitzler, and S. Stadtmüller. Radon - repair and diagnosis in ontology networks. In L. Aroyo, P. Traverso, F. Ciravegna, P. Cimiano, T. Heath, E. Hyvönen, R. Mizoguchi, E. Oren, M. Sabou, and E. P. B. Simperl, editors, *ESWC*, volume 5554 of *Lecture Notes in Computer Science*, pages 863–867. Springer, 2009.

[12] H. Knublauch, J. A. Hendler, and K. Idehen. SPIN - overview and motivation. W3C Member Submission, W3C, February 2011.

[13] D. Kontokostas, C. Bratsas, S. Auer, S. Hellmann, I. Antoniou, and G. Metakides. Internationalization of linked data: The case of the greek dbpedia edition. *Web Semantics: Science, Services and Agents on the World Wide Web*, 15(0):51 – 61, 2012.

[14] J. Lehmann, C. Bizer, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia - a crystallization point for the web of data. *Journal of Web Semantics*, 7(3):154–165, 2009.

[15] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 2013. Under review.

[16] P. N. Mendes, H. Mühleisen, and C. Bizer. Sieve: linked data quality assessment and fusion. In D. Srivastava and I. Ari, editors, *EDBT/ICDT Workshops*, pages 116–123. ACM, 2012.

[17] C. Rieß, N. Heino, S. Tramp, and S. Auer. EvoPat – Pattern-Based Evolution and Refactoring of RDF Knowledge Bases. In *Proceedings of the 9th International Semantic Web Conference (ISWC2010)*, Lecture Notes in Computer Science, Berlin / Heidelberg, 2010. Springer.

[18] E. Sirin and J. Tao. Towards integrity constraints in owl. In *Proceedings of the Workshop on OWL: Experiences and Directions, OWLED*, 2009.

[19] C. Stadler, J. Lehmann, K. Höffner, and S. Auer. Linkedgeodata: A core for a web of spatial open data. *Semantic Web Journal*, 3(4):333–354, 2012.

[20] O. Suominen and E. Hyvönen. Improving the quality of skos vocabularies with skosify. In *Proceedings of the 18th international conference on Knowledge Engineering and Knowledge Management*, EKAW'12, pages 383–397, Berlin, Heidelberg, 2012. Springer-Verlag.

[21] A. Zaveri, D. Kontokostas, M. A. Sherif, L. Bühmann, M. Morsey, S. Auer, and J. Lehmann. User-driven quality evaluation of dbpedia. In *To appear in Proceedings of 9th International Conference on Semantic Systems, I-SEMANTICS '13, Graz, Austria, September 4-6, 2013*. ACM, 2013.

[22] A. Zaveri, A. Rula, A. Maurino, R. Pietrobon, J. Lehmann, and S. Auer. Quality assessment methodologies for Linked Open Data. Submitted to the Semantic Web Journal.

[23] H. Zhu, P. A. V. Hall, and J. H. R. May. Software unit test coverage and adequacy. *ACM Comput. Surv.*, 29(4):366–427, 1997.

APPENDIX

Pattern Descriptions

In this section we present more detailed descriptions of Data Quality Test Patterns being part of our Test-driven Data Quality Methodology. We illustrate these patterns in detail and give examples of pattern bindings.

COMP Pattern. Depending on the property semantics, there are cases where two different literal values must have a specific ordering with respect to an operator. *P1* and *P2* are the datatype properties we need to compare and *OP* is the comparison operator R(*OP*) = `{ <, <=, >, >=, =, != }`.

```
SELECT ?s WHERE { ?s  %%P1%%  ?v1 .
                  ?s  %%P2%%  ?v2 .
                  FILTER ( ?v1 %%OP%% ?v2 ) }
```

Example bindings:
1. `dbo:deathDate` before '<' `dbo:birthDate`
2. `dbo:releaseDate` after '>' `dbo:latestReleaseDate`
3. `dbo:demolitionDate` before '<' `dbo:buildingStartDate`

MATCH Pattern. Application logic or real world constraints may put restrictions on the form of a literal value. *P1* is the property we need to check against *REGEX* and *NOP* can be a not operator ('!') or empty.

```
SELECT DISTINCT ?s WHERE { ?s  %%P1%%  ?value .
   FILTER ( %%NOP%% regex(str(?value), %%REGEX%) ) }
```

Example bindings:
1. `dbo:isbn` format is different '!' from "^([iIsSbBnN 0-9-])*$"
2. `dbo:postCode` format is different '!' from "^[0-9]{5}$"
3. `foaf:phone` contains any letters ("[A-Za-z]")

LITRAN Pattern. Application logic or real world facts may put restrictions on the range of a literal value depending on the type of a resource. *P1* is a property of an instance of class *T1* and its literal value must be between the range of [*Vmin*, *Vmax*] or outside (*NOP* can be a '!'). The query is phrased so that "between" does not require negation, but "outside" does.

```
SELECT DISTINCT ?s WHERE {
            ?s  rdf:type  %%T1%% .
            ?s  %%P1%%    ?value .
            FILTER( %%NOP%%
                    (?value < %%Vmin%% ||
                     ?value > %%Vmax%%))) }
```

Example bindings:
1. a `dbo:Person` should have `dbo:height` between 0.4 and 2.5 meters
2. the `geo:lat` of a `gml:_Feature` must be in range [-90,90]
3. the `geo:long` of a `gml:_Feature` must be in range [-180,180]

TYPEDEP Pattern. The type of a resource may imply the attribution of a second type. In this pattern *T1* and *T2* are the types tested for coexistence.

```
SELECT DISTINCT ?s WHERE {
        ?s  rdf:type  %%T1%% .
        FILTER NOT EXISTS { ?s rdf:type  %%T2%% } }
```

Example bindings:
1. `gml:_Feature` should imply `dbo:Place`
2. `yago:GeoclassCapitalOfAPoliticalEntity` should imply `dbo:Place`
3. `foaf:Person` should imply `dbo:Person`

TYPRODEP Pattern. Resources of a given type sometimes must be accompanied by a specified property. In this pattern the type *T1* is tested for coexistence with property *P1*.

```
SELECT DISTINCT ?s WHERE {
        ?s  rdf:type  %%T1%% .
        FILTER NOT EXISTS { ?s  %%P1%%  ?v } }
```

Example bindings: Resources representing
1. a `dbo:Place` should have a `geo:lat` property
2. a `dbo:Person` should have a `dbo:birthDate` property
3. a `dbo:Person` should have a `foaf:depiction` property

PVT Pattern. If a resource has a certain value *V* assigned via a property *P1* that in some way classifies this resource, one can assume the existence of other properties *P2*. The following pattern provides the test template for such cases.

```
SELECT DISTINCT ?s WHERE {
        ?s  %%P1%%  %%V1%%
        FILTER NOT EXISTS { ?s  %%P2%%  ?p } }
```

Example bindings: Resources
1. being extracted from a `dpt:Template:Geographic_location` should have a geo coordinate assigned (`dbo:georss:point`)
2. belonging to the category `dbc:1907_births` should have a `dbo:birthDate`
3. belonging to a Wikipedia category for maintenance, because they are using a template (`dbp:wikiPageUsesTemplate dbt:Infobox_character`), but have unlabeled fields (i.e. missing properties such as `dbpprop:first`)[30]

TRIPLE Pattern. In some cases hints with regards to errors or bad smells are already contained in the dataset. These are given as certain property *P1* value *V1* combinations and can be tested with the following pattern.

```
SELECT DISTINCT ?s WHERE { ?s  %%P1%%  %%V1%% }
```

Example bindings: Resources extracted from Wikipedia articles, that

[30] `http://en.wikipedia.org/wiki/Category:Articles_using_Infobox_character_with_multiple_unlabeled_fields`

1. were possibly copy-pasted (`dc:subject dbc:Possible_cut-and-paste_moves`)
2. have an inconsistent citation format (`dbp:wikiPageUsesTemplate dbt:Inconsistent_citations`)
3. have missing files (`dc:subject dbc:Articles_with_missing_files`)

ONELANG Pattern. A literal value should contain at most 1 literal for a language. *P1* is the property containing the literal and *V1* is the language we want to check.

```
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
```

Example bindings:
1. a single English ("en") `foaf:name`
2. a single English ("en") `rdfs:label`

RDFSDOMAIN Pattern. The attribution of a property is only valid when the class is in the domain of the property. In this pattern the property *P1* is tested for coexistence of the type *T1*. Optionally value *V1* can be specified to narrow the test to the specified value for *P1*.

```
SELECT DISTINCT ?s WHERE { ?s  %%P1%%  %%V1%% .
       FILTER NOT EXISTS {?s rdf:type ?T1 .
         ?T1 rdfs:subClassOf%%OP%% %%T1%% . }
       FILTER NOT EXISTS {?s rdf:type %%T1%%  } }
```

Example bindings:
1. `dc:subject dbc:CapitalsInAfrica` should have type `dbo:Place` attributed
2. `dbo:dissolved` should have type `dbo:SoccerClub` attributed

RDFSRANGE Pattern. The object of a triple must be within the range of the property. In this pattern property *P1* and type *T1* are tested for coexistence.

```
SELECT DISTINCT ?s WHERE { ?s  %%P1%%  ?c .
       FILTER NOT EXISTS {?c rdf:type ?T1 .
         ?T1 rdfs:subClassOf%%OP%% %%T1%% . }
       FILTER NOT EXISTS {?c rdf:type %%T1%%  } }
```

Example bindings:
1. the `dbo:spouse` of a `dbo:Person` must be a `dbo:Person`
2. the `dbo:birthPlace` of a `dbo:Person` must be a `dbo:Place`
3. the `dbo:dean` of a `dbo:EducationalInstitution` must be a `dbo:Person`

RDFSRANGED Pattern. The (literal) object of a triple must be of a certain datatype determined by the property used. In this pattern the property *P1* and the datatype *D1* are tested for coexistence.

```
SELECT DISTINCT ?s WHERE {
      ?s %%P1%% ?c.
      FILTER (DATATYPE(?c) != %%D1%%) }
```

Example bindings:
1. the value of the property `dbo:certificationDate` must be of type `xsd:date`
2. the value of the property `dbo:isPeerReviewed` must be of type `xsd:boolean`
3. the value of the property `dbo:successfulLaunches` must be of type `xsd:nonNegativeInteger`

INVFUNC Pattern. Some values assigned to a resource are considered to be unique for this particular resource and should not occur in connection with other resources. This pattern can be extended to also restrict the value as shown in the comments of the following listing.

```
SELECT DISTINCT ?s WHERE{
       ?a  %%P1%%  ?v1 . # ?a %%P2%%  %%V1%% .
       ?b  %%P1%%  ?v2 . # ?b %%P2%%  %%V1%% .
       FILTER ((str(?v1) == str(?v2)) && (?a != ?b))}
```

Example bindings:
1. two different resources should not have the same `foaf:homepage` (*P1*, *P2*)
2. two countries should not have the same `dbo:capital`

OWLCARD Pattern. Using this pattern, we can test for cardinal constraints on specific properties. *P1* is the property we need to compare with *V1* and *OP* is the comparison operator (`<`, `<=`, `>`, `>=`, `=`, `!=`)

```
SELECT DISTINCT ?s WHERE { ?s %%P1%% ?c }
GROUP BY ?s HAVING count(?c) %%OP%% %%V1%%
```

Example bindings:
1. every property defined as `owl:FunctionalProperty` (e.g. `dbo:birthDate`, `dbo:latestReleaseDate`) in the ontology cannot exist more than once (>1)
2. dbpedia.org's resources have an `rdfs:label` for each of its 20 different languages. Therefore each resource should not have more than 20 labels (>20), the same holds for other properties such as `rdfs:comment`.

OWLDISJC Pattern. A resource must not belong to two disjoint classes. *T1* and *T2* are the two disjoint classes we check.

```
SELECT DISTINCT ?s WHERE {
            ?s  rdf:type  %%T1%% .
            ?s  rdf:type  %%T2%% . }
```

Example bindings: (a) `dbo:Person` is `owl:disjointWith` with `dbo:Place`, (b) `dbo:Person` is `owl:disjointWith` with `dbo:Work`,

OWLDISJP Pattern. A triple object v cannot be assigned to a resource s via both properties *P1* and *P2* if these are stated to be disjoint by an `owl:disjointProperty` axiom.

```
SELECT DISTINCT ?s WHERE { ?s %%P1%% ?v .
                           ?s %%P2%% ?v .}
```

Example bindings:

1. `skos:prefLabel` is disjoint with `skos:hiddenLabel`
2. `dbo:bandMember` is disjoint with `dbo:birthPlace`

OWLASYMP Pattern. For a given property *P1* that is declared to be asymmetric, this pattern checks if there are violating cases where it is nonetheless used as symmetric property, i.e. for two resources *a* and *b* there are axioms for *a P1 b.* and *b P1 a.* .

```
SELECT ?r1 WHERE { ?r1 %%P1%% ?r2 .
                   ?r2 %%P1%% ?r1 . }
```

Example bindings:

1. child parent relations (`dbo:child`) cannot be symmetric
2. person birth place relations (`dbo:birthPlace`) cannot be symmetric

OWLIRREFL Pattern. For a given property *P1* that is declared to be irreflexive, this pattern find violating statements that nonetheless use this property reflexively, i.e. for a resource *a* there is an axiom *a P1 a.* .

```
SELECT DISTINCT ?s WHERE {?s %%P1%% ?s .}
```

Example bindings:

1. a resource cannot be its own parent (`dbo:parent`)
2. a resource cannot be its own child (`dbo:child`)