Test-driven Data Quality Evaluation for SPARQL Endpoints

Dimitris Kontokostas[1], Sören Auer[1], Sebastian Hellmann[1], Jens Lehmann[1], Patrick Westphal[1], Roland Cornelissen[2], and Amrapali Zaveri[1]

[1] University of Leipzig, Institute of Computer Science, AKSW Group, Augustusplatz 10, D-04009 Leipzig, Germany
{lastname}@informatik.uni-leipzig.de, http://aksw.org
[2] Metamatter, Den Horn, Netherlands
roland@metamatter.nl, http://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. In this paper we, present a methodology for test-driven data quality assessment, which is inspired by test-driven software development. We argue, that 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 literature review, we compile a comprehensive library of quality test patterns. The main contribution of our work is an extensive, unprecedented evaluation of DBpedia data quality employing our test methodology. 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.

Keywords: Data Quality, Linked Data, DBpedia

1 Introduction

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 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.

In this paper we, present a methodology for test-driven data quality assessment, which is inspired by test-driven software development. Compared to

software source code testing, where test cases have to be implemented largely manually or with limited programmatic support, the situation in knowledge base testing on the Semantic Web is slightly more advantageous. On the Semantic 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 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 literature review, we compiled a comprehensive library of quality tests. The main contribution of our work is the extensive evaluation of DBpedia data quality employing our test methodology. Another main advantage 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. Further contributions are as follows:

- From our test cases, we generalized a pattern library which can be instantiated for rapid development of more test cases.
- We list and discuss actual errors found in DBpedia. We also succeed in quantifying these errors.
- We present a lightweight formalism for instantiating standard test cases.
- Our framework is easily re-usable for other knowledge bases and triple store implementations as we built upon the SPARQL 1.1 standard.

The paper is structured as follows: Section 2 describes the three-step methodology we followed to define *Data Quality Patterns*. While the creation of our initial pattern library is described in Section 3, the resulting Pattern Library is elaborated in Section 4. We instantiate, run and evaluate the tests in Section 5, followed by a discussion in Section 6. Section 7 elaborates on related work and we conclude in Section 8.

2 Test-driven Data Quality Methodology

Our methodology (cf. Figure 1) consists of the following major steps:

1. *Definition of data quality test patterns.* Those define abstract classes of potential data quality problems, which we collected in a library in the subsequent section. These patterns were generalized from incidental reports about data quality problems (cf. Section 4 for details).
2. *Instantiation of data quality test patterns.* Abstract test patterns are instantiated for concrete vocabularies and knowledge bases with bindings.
3. *Execution of the resulting data quality test cases.* The test results serve as input for a data quality improvement activity. The tests can be run repeatedly to verify the effectiveness and assist in data debugging.

The first crucial step is the definition of a data quality test pattern (DQTP).

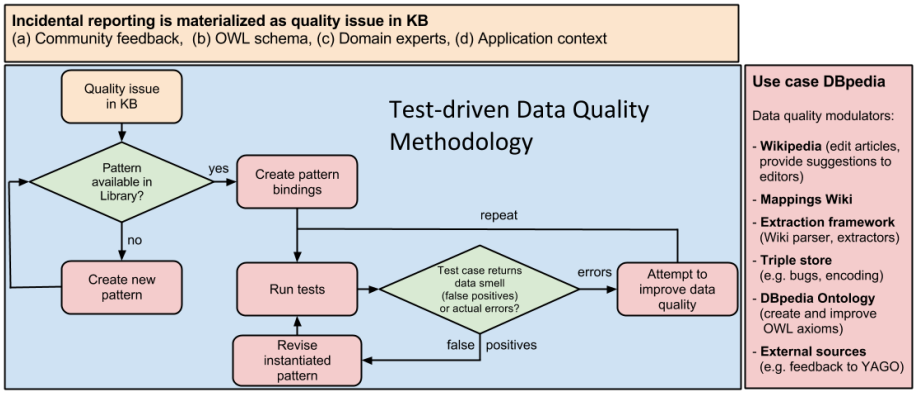

Fig. 1. Flowchart showing the test-driven data quality methodology (middle part). The top serves as input for the methodology, the right part displays concrete leverage points for improvements in the DBpedia use case.

Definition 1 (Data Quality Test Pattern). *A data quality test pattern is a tuple (V, S), where V is a set of typed variables, S is a SPARQL query template with placeholders for the variables from V. Possible types of the variables are IRIs, literals, operators, datatype values (e.g. integers) and regular expressions. With $R(v)$ we denote the value range for a variable in $v \in V$ and with $R(V)$ the union of all these sets, i.e. $R(V) = \bigcup_{v \in V} R(v)$.*

DQPTs are knowledge base and vocabulary agnostic. An example DQTP is:

```
1 SELECT ?s WHERE { ?s  %%P1%%   ?v1 .
2                   ?s  %%P2%%   ?v2 .
3                   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.

Definition 2 (Test Pattern Binding). *A test pattern binding for a data quality test pattern (V, S, C) is a mapping from V to the set of value ranges for the variables in V, i.e. $V \rightarrow R(V)$, where each $v \in V$ is mapped to a suitable element of its value range $R(v)$. In addition, each test pattern binding has an associated data quality issue type $C \in \{error, bad\_smell\}$.*

Definition 3 (Data Quality Test Cases). *A data quality test case is the result of the application of a test pattern binding to a DQTP.*

An example test pattern binding and resulting data quality test case is[3]:

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

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

3 Pattern Elicitation and Creation

We performed three different analyses for elicitation of 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. Analysis of the ontology schema of the DBpedia OWL ontology.

Community feedback. We thoroughly reviewed all the DBpedia related mailing lists and QA websites, that is *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 categories. The maintenance of the immense amount of content in Wikipedia poses a severe challenge to the core of the Wikipedia editors. Seasoned Wikipedians (e.g. admins and stewards) therefore use special *Categories* and *Templates* for administrating and tagging errors in the article space[8]. We analyzed these categories and created patterns to assign errors automatically to capture them in our assessment. Although all Wikipedia categories were extracted with the DBpedia extraction framework, all templates were not. Thus, we developed a new DBpedia extractor that extracts all templates used inside an article and loaded the template data into the main DBpedia endpoint. Our TRIPLE Pattern can now recreate most of the Wikipedia maintenance pages. In addition, we used the PVT Pattern (see binding example c) to simulate and automatize some of the maintenance tags. Very often much manual work is required to assign these tags.

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

[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

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 (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.

4 Pattern Library

Our *Pattern Library* consists of 14 patterns. Table 1 shows a description of all patterns along with two example bindings. In the following, we describe each pattern in detail.

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: (a) `dbo:deathDate` before '<' `dbo:birthDate`, (b) `dbo:releaseDate` after '>' `dbo:latestReleaseDate` (c) `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 ?s WHERE { ?s  %%P1%%  ?value .
                  FILTER ( %%NOP%% regex(str(?value), %%REGEX%%) ) }
```

Example bindings: (a) `dbo:isbn` format is different '!' from "^([iIsSbBnN 0-9-])*$" (b) `dbo:postCode` format is different '!' from "^[0-9]{5}$", (c) resources having a yago type (`rdf:type`) with more than one consecutive capital letters ("^http://dbpedia.org/class/yago/.*[a-z][A-Z][A-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.

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

Template	Binding example	Src
COMP: Comparison between two literal values of a resource	a. `dbo:deathDate` after `dbo:birthDate`	CF
	b. `dbo:releaseDate` after `dbo:latestReleaseDate`	CF
MATCH: Literal matches a pattern	a. wrong `dbo:isbn` number	CF
	b. wrong `dbo:postcode` format	CF
LITRAN: Literal in specific range	a. height of a `dbo:Person` in [0.4,2.5]	CF
	b. geographic latitude in [-90,90] (WGS 84)	CF
CARD: Cardinality restriction on property	a. `dbo:birthDate` is a functional property	DO
	b. there should not be more than 20 `rdfs:label` in dbpedia.org	CF
TYPEDEP: Type dependency (resource is of type t_1 and not of type t_2)	a. has coordinates (`gml:_Feature`) but is not a `dbo:Place`	CF
	b. is a `yago:GeoclassCapitalOfAPoliticalEntity` but not a `dbo:Place`	CF
PROPDEP: Resource has property p_1 but not property p_2	a. has `dbo:deathDate` but not `dbo:birthDate`	CF
	b. has `dbpprop:dateOfBirth` but not `dbo:birthDate`	CF
PDOMAIN: Resource has property p_1 but is not of type t_1	a. resource cannot have a `foaf:name` property if it is not an `owl:Thing`	CF
	b. has the "Cities of Africa" category but does not have the type `dbo:City`	CF
PRANGE: Object of a triple is not of type t_1	a. a `dbo:Person`'s spouse must be a `dbo:Person`	DO
	b. `dbo:birthPlace` of a `dbo:Person` must be a `dbo:Place`	DO
TYPRODEP: Resource of type t_1 should contain property p_1	a. a `dbo:Place` should have coordinates	CF
	b. a `dbo:Person` should have a `dbo:birthDate`	CF
PVT: Resources with a property p_1 having value v_1 should contain a triple with property p_2	a. articles with a Geographic_location template should extract coordinates	CF
	b. resources in the category of "1907 births" should have a `dbo:birthdate`	W
TRIPLE: Resource has value v_1 for property p_1	a. Wikipedia articles that are possibly copy-pasted	W
	b. Wikipedia articles with inconsistent citation format	W
INVFUNC: Unique value constraint	a. two resources with the same `foaf:homepage`	DO
	b. two countries with the same capital	DO
DISJOIN: Disjoint class constraint	a. a `dbo:Person` is disjoint with `dbo:Place`	DO
	b. `dbo:Person` is disjoint with `dbo:Work`	DO
ONELANG: One literal for a language constraint	a. a single English `foaf:name`	CF
	b. a single English `rdfs:label`	CF

Table 1. Example templates and bindings. The column *Src* contains the source where the corresponding pattern is derived from, i.e. if it is created due to community feedback (CF), utilizing Wikipedia maintenance categories (W) or analyzing the ontology schema of the DBpedia OWL ontology (DO) as described in Section 3

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

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

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

Example bindings: (a) every property defined as `owl:FunctionalProperty` (e.g. `dbo:birthDate`, `dbo:latestReleaseDate`) in the ontology cannot exist more than once (>1), (b) 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`.

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: (a) `gml:_Feature` should imply `dbo:Place`,
(b) `yago:GeoclassCapitalOfAPoliticalEntity` should imply `dbo:Place`, (c) `foaf:Person` should imply `dbo:Person`.

PROPDEP Pattern: Certain properties imply the attribution of a second property. In this pattern *P1* and *P2* are the properties tested for coexistence.

```
SELECT ?s WHERE { ?s  %%P1%%  ?v1 .
                  FILTER NOT EXISTS { ?s  %%P2%%  ?v2 } }
```

Example bindings: A resource with property (a) `dbo:deathDate` should have a property `dbo:birthDate`, (b) `dbp:dateOfBirth` should have a property `dbo:birthDate`, (c) `dbo:activeYearsStartDate` should have a property `dbo:activeYearsEndDate`.

PDOMAIN 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%% } }
```

Example bindings: (a) `foaf:name` should have type `owl:Thing` attributed, (b) `dc:subject dbc:CapitalsInAfrica` should have type `dbo:Place` attributed, (c) `dbo:dissolved` should have type `dbo:SoccerClub` attributed.

PRANGE 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 ?c WHERE { ?s  %%P1%%  ?c .
                  FILTER NOT EXISTS { ?c  rdf:type  %%T1%% } }
```

Example bindings: (a) the `dbo:spouse` of a `dbo:Person` must be a `dbo:Person`, (b) the `dbo:birthPlace` of a `dbo:Person` must be a `dbo:Place`, (c) the `dbo:dean` of a `dbo:EducationalInstitution` must be a `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 * WHERE { ?s  rdf:type  %%T1%% .
                 FILTER NOT EXISTS { ?s  %%P1%%  ?v } }
```

Example bindings: Resources representing (a) a `dbo:Place` should have a `geo:lat` property. (b) a `dbo:Person` should have a `dbo:birthDate` property, (c) 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*. This pattern is a generalization of the PROPDEP Pattern.

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

Example bindings: resources (a) being extracted from a `dpt:Template:Geographic_location` should have a geo coordinate assigned (`dbo:georss:point`), (b) belonging to the category `dbc:1907_births` should have a `dbo:birthDate`, (c) 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`)[10]

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

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

Example bindings: Resources extracted from Wikipedia articles, that (a) were possibly copy-pasted (`dc:subject dbc:Possible_cut-and-paste_moves`), (b) have an inconsistent citation format (`dbp:wikiPageUsesTemplate dbt:Inconsistent_citations`), (c) have missing files (`dc:subject dbc:Articles_with_missing_files`).

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. See Section 6 for comments.

```
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: (a) Two different resources should not have the same `foaf:homepage` (P1, P2), or (b) two countries the same `dbo:capital`.

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

```
SELECT ?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`,

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 ?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: (a) a single English ("en") `foaf:name`, (b) a single English ("en") `rdfs:label`.

5 DBpedia Data Quality Evaluation[11]

Recent developments in DBpedia internationalization [8] resulted in the creation of new DBpedia language chapters. DBpedia chapters extract data from a respective Wikipedia language edition and load the extracted data on a separate domain (e.g. `http://nl.dbpedia.org` for Dutch). Although every chapter uses its own namespace for resources (e.g. `http://nl.dbpedia.org/resource/`) all chapters use the same DBpedia ontology. This facilitates cross-DBpedia

[11] To reproduce the evaluation, we published all scripts, queries and data used to generate the tables and images on `https://github.com/AKSW/Databugger`.

Language	Triples	Subjects	Errors	Error rate
Czech	20,585,673	379,311	40,187	13.96%
Dutch	59,049,164	1,496,453	261,455	8.90%
Dutch (L)	56,762,114	1,465,212	366,893	8.08%
English	329,267,934	9,426,357	3,598,765	19.19%
English (L)	282,656,622	9,833,222	2,686,854	17.38%
French	92,932,838	2,455,021	476,985	13.06%
German	95,320,757	2,378,952	1,111,239	10.97%
Greek	6,747,265	107,333	28,699	17.23%
Japanese	46,358,390	858,529	180,945	16.91%
Spanish	83,909,360	2,215,745	474,566	17.77%
Total			9,226,588	14.35%

Table 2. Evaluation summary for all DBpedia language editions. *Triples* refers to total number of triples in the triple store and *Subjects* to unique DBpedia resources. *Errors* are the total errors encountered and *Error Rate* the average error rate for all DQTC.

SPARQL query interoperability. A complete list of all the available DBpedia language chapters can be found on the DBpedia website[12], however, not all chapters provide a high level of online availability. For this evaluation we used the Czech, Dutch, English, French, German, Greek and Japanese DBpedia language chapters along with two DBpedia Live editions [12]: the English and the Dutch. [13].

To evaluate our methodology all the instantiated bindings (cf. Table 1) were executed against the SPARQL endpoints of all aforementioned DBpedia language chapters. Most of the bindings are based on the DBpedia ontology or common vocabularies and thus, can be run on all endpoints. However, not all endpoints are as complete as the English one (i.e. mappings of local properties to the DBpedia ontology are incomplete) which lowers the number of candidate errors on non-English endpoints. Moreover, bindings PROPDEP-c, PDOMAIN-b and all bindings of the TRIPLE pattern are specific to the English DBpedia.

Table 2 provides an evaluation summary for all endpoints. The first two columns (*Triple Count* and *Unique Subjects*) give an estimate of the size of the evaluated DBpedia language edition. *Total Errors* is the sum of the result of all Data Quality Test Cases (DQTC) and *Average Error Rate* is the average error rate of all DQTCs for a language. Also note that these results contain all the instantiated DQTCs and not only the first two as in Table 3. The execution time of all DQTC on all endpoints was 72 minutes.

A depiction of four high error rate DQTCs is available in Figure 2. Images (a), (b) and (c) depict the absolute errors, the prevalence and the error rate for MATCH-a, TYPEDEP-a, PRANGE-b and TYPRODEP-a respectively. Moreover, image (d) depicts the total execution time separately for each endpoint.

[12] `http://wiki.dbpedia.org/Internationalization/Chapters`

[13] In order of appearance: `http://cz.dbpedia.org`, `http://nl.dbpedia.org`, `http://dbpedia.org`, `http://fr.dbpedia.org`, `http://de.dbpedia.org`, `http://el.dbpedia.org`, `http://ja.dbpedia.org`, `http://live.dbpedia.org`, `http://live.nl.dbpedia.org`

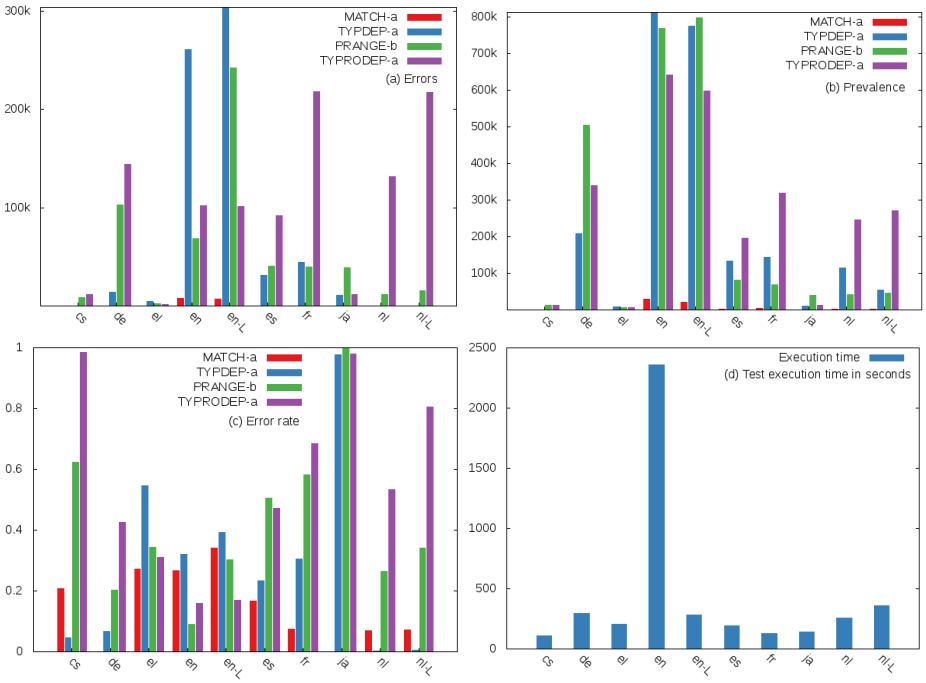

Fig. 2. Histograms of four high error rate DQTCs: (a), (b) and (c) depict the absolute errors, the prevalence and the error rate for MATCH-a, TYPEDEP-a, PRANGE-b and TYPRODEP-a respectively; (d) depicts the total execution time for each endpoint.

Finally, in accordance with the bindings of Table 1, we present a complete evaluation of the the English DBpedia and English DBpedia Live in Table 3. Prevalence is calculated by inserting the bindings in an a related companion template for prevalence. For the TRIPLE patterns, the DQTCs are SPARQL queries with a single triple pattern and thus prevalence cannot be calculated. Furthermore, TRIPLE-b dependes on the new *Template extractor* (cf. Section 3) that was not loaded on the endpoints at the time of the evaluation.

6 Discussion

One of the most surprising fact was, that we could actually find one of the largest error with the PDOMAIN Pattern. Missing explicated domain types for resources were very frequent among incident reports by the community. Writing inferred domain types into the triple store happens in several different places in the extraction software and is very hard to track, so that this error class was only partially fixed in an unsystematic way and then re-occurred again in consecutive DBpedia versions. Our approach helps to ensure sustainable improvement of DBpedia, as it allows to discover re-occurring errors automatically without a

Binding	DBpedia English			DBpedia English Live		
	Error	Prevalence	Error rate	Error	Prevalence	Error rate
CARD-a	390,661	610,202	64.02%	0	809,799	0.00%
CARD-b	5,009	23,891,055	0.02%	0	10,889,643	0.00%
COMP-a	757	226,024	0.33%	359	410,279	0.09%
COMP-b	2	118	1.69%	10	204	4.90%
DISJOIN-a	86	24,424,883	0.00%	47	14,654,413	0.00%
DISJOIN-b	0	24,424,883	0.00%	41	14,654,412	0.00%
INVFUNC-a	68,034	578,169	11.76%	86,025	545,445	15.77%
INVFUNC-b	1,530	3,886	39.37%	931	1,961	47.47%
LITRAN-a	85	44,267	0.19%	4,988	56,475	8.83%
LITRAN-b	0	758,316	0.00%	0	841,913	0.00%
MATCH-a	7,553	28,256	26.73%	6,975	20,470	34.07%
MATCH-b	63,224	156,190	40.48%	66,962	165,681	40.42%
ONELANG-a	0	2,576,741	0.00%	0	2,536,228	0.00%
ONELANG-a	0	24,841,480	0.00%	0	10,889,644	0.00%
PDOMAIN-a	502,623	2,267,864	22.16%	512,384	2,203,975	23.25%
PDOMAIN-b	45	63	71.43%	44	62	70.97%
PRANGE-a	6,898	20,124	34.28%	11,172	21975	50.84%
PRANGE-b	68,671	768,759	8.93%	241,935	799,160	30.27%
PROPDEP-a	54,195	255,465	21.21%	40,364	328,829	12.28%
PROPDEP-b	241,556	829,941	29.11%	247,471	1,046,851	23.64%
PVT-a	4,437	12,824	34.60%	539	5,885	9.16%
PVT-b	1,055	3,735	28.25%	1,189	3,813	31.18%
TRIPLE-a	1,804	-	0.00%	1,808	-	0.00%
TRIPLE-b	0	-	0.00%	0	-	0.00%
TYPDEP-a	261,173	813,153	32.12%	303,782	776,049	39.14%
TYPDEP-b	16	150	10.67%	0	0	0.00%
TYPRODEP-a	101,677	642,632	15.82%	101,243	597,832	16.94%
TYPRODEP-b	392,515	763,644	51.40%	188,760	572,398	32.98%

Table 3. Detailed statistics for all DQTCs executed in English DBpedia and English DBpedia Live.

resource-intensive community reporting loop. Having `owl:Thing` (PDOMAIN-a) manifested in the triple store seems trivial, but is a requirement (e.g. for OWL-DL) of quite a few tools in the SW tool chain and can easily break existing applications. A DBpedia developer familiar with the software is easily able to quickly spot the error source based on the property binding (e.g. `foaf:name` in this case).

Revision of instantiated patterns. Although our pattern library already covers a wide range of data quality errors in DBpedia, there are cases where the mere instantiation of patterns is not sufficient. Binding COMP-a (cf. Table 1), for example, returns 757 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 thus, a false positive. An improvement of this particular test case might look like:

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

Another similar case is MATCH-c, where YAGO types contained spelling errors in resources (two consecutive capital letters). The false positives in this case are e.g. resources containing the substring "BC"(as in before Christ). We reduced the number of false positives by searching YAGO types for 'BC' and creating a white list, which we included in the test case. Based on this, we included the extra false-positive-reduction loop for expert users at the bottom of Figure 1.

The **INVFUNC Pattern** was originally brought up by a community member who reported duplicate `dbo:capital` usages. As a consequence, we initially created the template as a straightforward conversion of the semantics of `owl:InverseFunctionalProperty`, but reinterpreted with SPARQL and the Unique Name Assumption. As we started to run the tests, however, only false positives where returned for `foaf:homepage` and `dbo:capital`, e.g. Canberra is the capital of Australia and the Australian Capital Territory and `http://www.eurovision.tv/` is the homepage for each yearly edition of the Eurovision Song Contest as well as of its abstract entity. A direct mapping from OWL was therefore not useful and we will in the future restrict the pattern to a single facet by adding: ?a %%P1%% %%V1%% . ?b %%P1%% %%V1%% . Our conclusion is that `owl:InverseFunctionalProperty` should only be assigned, if the meaning of the property has a very specific domain.

While axioms in an OWL scheme are intended to be applicable in a global context, our test-driven methodology clearly depends on local requirements. Data constraints, for example, can be very application specific and are not universally valid. For instance, due to the vast size of DBpedia, it is unrealistic to expect data completeness, e.g. that every `dbo:Person` has a `foaf:depiction` and a `dbo:birthDate` (TYPRODEP pattern). However, in the context of an application like "A day like today in history"[14] these properties are expected, as they lead to a failure. Thus, the definition of an *error* or a *bad_smell* is subjective to the binding creator's requirements.

7 Related Work

Previous Data Quality Measurements on DBpedia. An earlier publication [1] concerning the data quality of extracted RDF triples mainly concentrates on the data source, the 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.

[14] `http://el.dbpedia.org/apps/DayLikeToday/`

In [10] 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 [10] is the fusion of cross-language knowledge of the language specific DBpedia instances. This topic as well as other internationalization issues are treated in [8]. There, different extraction problems of the Greek DBpedia are presented that can also be applied to other languages, especially those using non-Latin characters. These are e.g. the use of special templates, URI and IRI encoding and the cross-language linkage of DBpedia resources. The authors' approach to counter these issues and increase the data quality is to expand the extraction framework and introduce language-specific *i18n filters*.

Another study aimed to develop a framework for the DBpedia quality assessment is presented in [9]. In this study, particular problems of the DBpedia extraction framework were taken into account and integrated in the framework. However, only a small sample (75 resources) was assessed in this case and an older DBpedia version (2010) was analyzed.

General Data Quality Assessment. There exist several approaches towards developing frameworks in order to assess the data quality of LOD. We only give a brief overview here and refer to [15] for details. These frameworks can be broadly classified into (i) automated (e.g. [5]), (ii) semi-automated (e.g. [4]) or (iii) manual (e.g. [2,11]) 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 considerable amount of user 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 [3] 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 [6], 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 [13].

Rules and SPARQL. *SPARQL Inferencing Notation* (SPIN) [7] is W3C submission aiming at representing rules and constraints on Semantic Web mod-

els. 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 our *Pattern Bindings*. SPIN function arguments must have specific constraints on the argument datatype or argument class and do not support operators (e.g. '=', '>', '!'). Four of our templates above cannot be directly represented in SPIN. 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. One of the advantages of converting our templates to SPIN is that they can be stored directly in RDF. However, a disadvantage is that SPIN is more complex than the lightweight framework proposed by us. From the efforts related to SPIN, we re-used the data quality pattern library as well as the ontologies.

Another related approach is the Pellet Integrity Constraint Validator (ICV)[15]. *Pellet ICV* [14] translates OWL integrity constraints into SPARQL queries. Similar to our approach, the execution of those SPARQL queries indicates violations. An implication of this, is that the unique names assumption and a closed world assumption is in effect. The difference to our approach is that Pellet ICV directly works on OWL files without an explicit representation of constraints and the translation happens behind the scenes. In this sense, an OWL representation could be seen as a different representation mechanism for which the end result after employing Pellet ICV is similar to the template bindings in our approach. A difference is, of course, that there is no pre-defined template mechanism comparable to our library above in OWL.

Schemarama[16] 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 problem.

For XML, *Schematron*[17] 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 extensive testing of DBpedia data quality. Our evaluation on a number of language specific DBpedia endpoints showed, that DQTPs are able to reveal a substantial amount of data quality issues in an effective and efficient way.

[15] http://clarkparsia.com/pellet/icv/
[16] http://swordfish.rdfweb.org/discovery/2001/01/schemarama/
[17] http://www.schematron.com/

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 actually fix problems. 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 on of the most pressing problems of the Data Web – the improvement of data quality and the increase of Linked Data fitness for use.

Acknowledgment

We would like to thank the active members of the DBpedia community and in particular the chapter activists, without whom this work would not have been possible. This work was supported by a grant from the European Union's 7th Framework Programme provided for the project LOD2 (GA no. 257943).

References

1. S. Auer and J. Lehmann. What have Innsbruck and Leipzig in common? extracting semantics from wiki content. In *ESWC*, volume 4519 of *LNCS*, pages 503–517. Springer, 2007.
2. C. Bizer and R. Cyganiak. Quality-driven information filtering using the WIQA policy framework. *Web Semantics*, 7(1):1 – 10, Jan 2009.
3. 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.
4. A. Flemming. Quality characteristics of linked data publishing datasources. Master's thesis, Humboldt-Universität of Berlin, 2010.
5. C. Guéret, P. T. Groth, C. Stadler, and J. Lehmann. Assessing linked data mappings using network measures. In *ESWC*, volume 7295 of *LNCS*, pages 87–102. Springer, 2012.
6. A. Hogan, A. Harth, A. Passant, S. Decker, and A. Polleres. Weaving the pedantic web. In *LDOW*, 2010.
7. H. Knublauch, J. A. Hendler, and K. Idehen. Spin - overview and motivation. W3C Member Submission, W3C, February 2011.
8. D. Kontokostas, C. Bratsas, S. Auer, S. Hellmann, I. Antoniou, and G. Metakides. Internationalization of linked data: The case of the greek dbpedia edition. *JWS*, 15(0):51 – 61, 2012.
9. P. Kreis. Design of a quality assessment framework for the dbpedia knowledge base. Master's thesis, Freie Universität Berlin, 2011.
10. J. Lehmann, C. Bizer, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia - a crystallization point for the web of data. *JWS*, 7(3):154–165, 2009.
11. B. C. Mendes P.N., Mühleisen H. Sieve: Linked data quality assessment and fusion. In *LWDM*, 2012.
12. M. Morsey, J. Lehmann, S. Auer, C. Stadler, and S. Hellmann. DBpedia and the Live Extraction of Structured Data from Wikipedia. *Program: electronic library and information systems*, 46:27, 2012.

13. C. Rieß, N. Heino, S. Tramp, and S. Auer. EvoPat – Pattern-Based Evolution and Refactoring of RDF Knowledge Bases. In *ISWC2010*, LNCS. Springer, 2010.
14. E. Sirin and J. Tao. Towards integrity constraints in owl. In *Proceedings of the Workshop on OWL: Experiences and Directions, OWLED*, 2009.
15. A. Zaveri, A. Rula, A. Maurino, R. Pietrobon, J. Lehmann, and S. Auer. Quality assessment methodologies for Linked Open Data. Submitted to SWJ.