Databugger: A Test-Driven Framework for Debugging the Web of Data

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 Databugger, a framework for test-driven quality assessment of Linked Data, which is inspired by test-driven software development. Databugger ensures a basic level of quality by accompanying vocabularies, ontologies and knowledge bases with a number of test cases. The formalization behind the tool employs SPARQL query templates, which are instantiated into concrete quality test queries. The test queries can be instantiated automatically based on a vocabulary or manually based on the data semantics. 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, Web of Data, Linked Data, SPARQL

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 [4]. 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 [1] or (2) automated, reasoning based methods [3].

In this demo, we present Databugger, a framework for test-driven Linked Data quality assessment, which is inspired by test-driven software development. 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. Databugger exploits the RDF data model by devising a pattern-based approach for the data quality tests of knowledge bases. Ontologies, vocabularies and knowledge bases can be accompanied by a number of test cases, which help to ensure a basic level of quality. This is achieved by employing SPARQL query templates, which are instantiated into concrete quality test SPARQL queries. We provide a comprehensive library of quality test patterns, which can be instantiated for rapid development of more test cases. 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.

In this demo, we will showcase how Databugger can be used for Linked Data quality assurance. In particular, we describe the *Test-Driven Quality Evaluation* methodology (Section 2). We outline the architecture and extensibility of Databugger (Section 3) and provide an overview of use case evaluations with five different datasets (Section 4). We con-

WWW'14 Companion, April 7–11, 2014, Seoul, Korea.
ACM 978-1-4503-2745-9/14/04.
todo: http://DOI string/url.

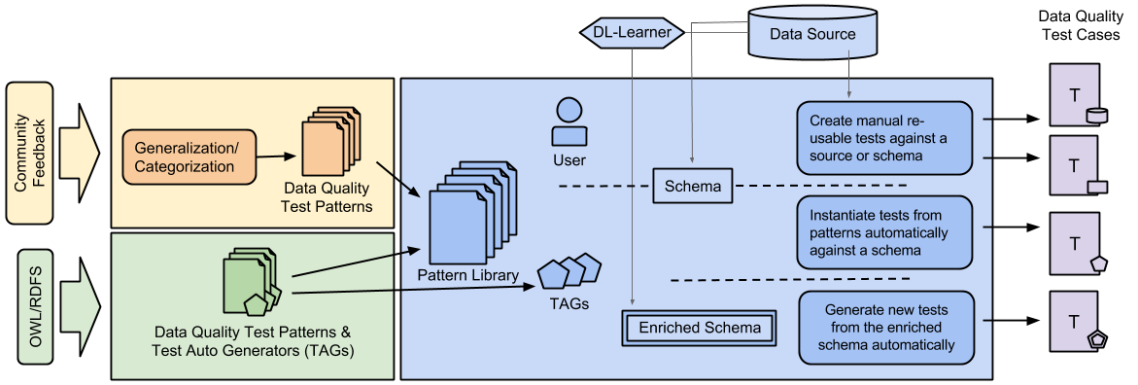

Figure 1: Flowchart showing the test-driven data quality methodology. The left part displays the input sources of our pattern library. In the middle part the different ways of pattern instantiation are shown which lead to the Data Quality Test Cases on the right.

clude and point to possible future extensions of Databugger (Section 5).

2. TEST-DRIVEN QUALITY ASSESSMENT IN A NUTSHELL

In this section we introduce basic notions of our methodology and describe the workflow implemented in the Databugger tool. A thorough description of test-driven quality assessment methodology can be found in [5][1]

Data Quality Test Pattern (DQTP). A data quality test pattern is a SPARQL query template with variable placeholders. Possible types of the pattern variables are IRIs, literals, operators, datatype values (e.g. integers) and regular expressions. Using **%%v%%** as syntax for placeholders, an example DQTP is:

```
1 SELECT ?s WHERE { ?s %%P1%% ?v1. ?s  %%P2%%   ?v2 .
2                   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. Test pattern bindings are valid DQTP variable replacements.

Data Quality Test Case. Applying a pattern binding to a DQTP results in an executable SPARQL query. Each result of the query is considered to be a violation of a test case. A test case may have three different results: success (empty result), violation (results are returned) and timeout (test is marked for further inspection). An example test pattern binding and resulting data quality test case is[2]:

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

Test Auto Generator (TAG). A Test Auto Generator reuses the RDFS and OWL modelling of a knowledge base to verify data quality. In particular, a TAG, based on a DQTP, takes a schema as input and returns test cases. TAGs consist of a detection and an execution part. The detection part is a query against a schema and for every result of a detection query, a test case is instantiated from the respective pattern, for instance:

```
1 # TAG                                  | # TQDP
2 SELECT DISTINCT ?P1 ?P2                | SELECT DISTINCT
3 WHERE {                                | ?s WHERE {
4 ?P1 owl:propertyDisjointWith ?P2.      | ?s %%P1%% ?v.
5 }                                      | ?s %%P2%% ?v.}
```

Additionally, 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) [5].

The test-driven data quality methodology is illustrated in Figure 1. As shown in the figure, there are two major sources for the creation of 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 to create tests:

1. *Manually create test cases:* Test cases specific to a certain dataset or schema can be written manually by an engineer. This can be guided choosing suitable DQTPs of our proposed pattern library. Tests that refer to the schema of a common vocabulary can become part of a central library to facilitate later reuse.

2. *Reusing tests based on common vocabularies:* Naturally, a major goal in the Semantic Web is to reuse existing vocabularies instead of creating new ones. We detect the used vocabularies in a dataset, which allows to reuse tests from a library, currently delivered with the Databugger tool.

3. *Using `RDFS/OWL` constraints directly:* As previously explained, tests can be automatically created via TAGs in this case.

4. *Enriching the `RDFS/OWL` constraints:* Since many datasets provide only limited schema information, we perform automatic schema enrichment as recently researched in [2]. Those schema enrichment methods can

[1] `http://svn.aksw.org/papers/2014/WWW_Databugger/public.pdf`

[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`

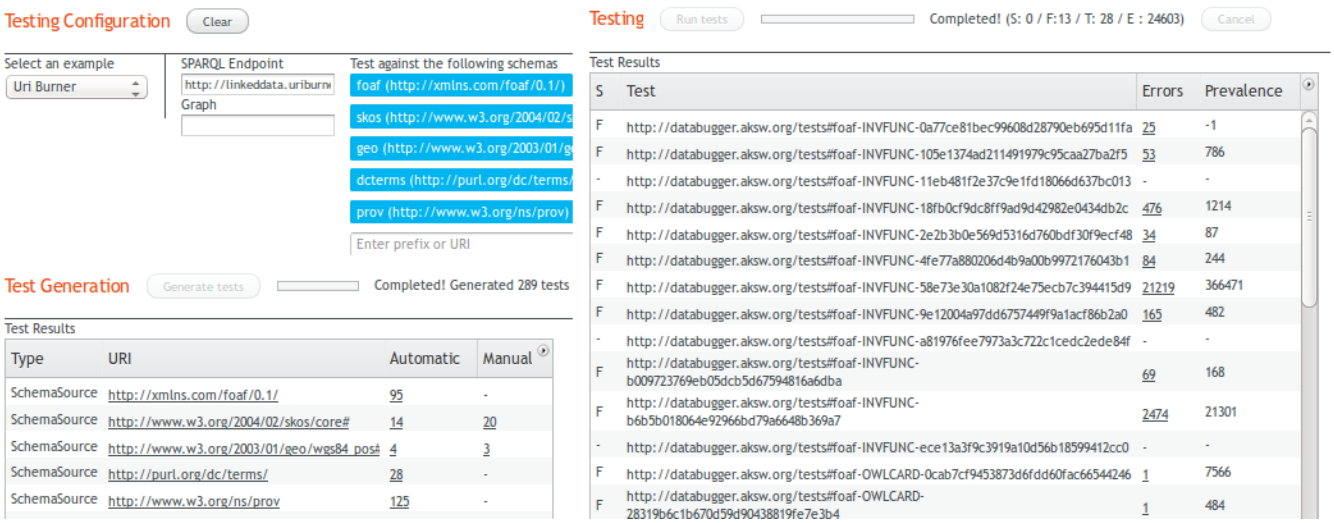

Figure 2: Screenshot of the Databugger web interface. In the upper left section the user configures the SPARQL Endpoint, the graph and the schemas she wants to test her data with. In the lower left section, test cases are automatically generated by parsing the schemas. Manual tests predefined for a schema are also loaded. In the right section Databugger starts running the tests and displays test results to the user.

take an RDF 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.

3. TOOL OVERVIEW, ARCHITECTURE AND EXTENSIBILITY

Databugger is a tool built to showcase the *test-driven quality assessment methodology*. The tool is released as open source under the Apache License and provides both, a command line interface (CLI) and a web interface (cf. Figure 2)[3,4]. A simple CLI test configuration can be generated with:

```
$ databugger -d <dataset-uri> -e <endpoint-uri>
             -g <graph1|graph2|...>
             -s <schema-prefix1,schema-prefix2,...>
```

Once the user starts a test configuration, the framework dereferences and reads the schemas. For all the provided schemas, Databugger generates automatic test cases using TAGs and loads any existing manual test cases for the schemas or the dataset. All the test cases are then executed against the SPARQL endpoint. The test results are both displayed on the screen and stored in RDF. The web interface allows the user to generate the same test configuration from a more interactive interface (cf. Figure 2).

The DQTPs[5], the manual and auto-generated test cases[6], the TAGs[7] and the test results are modeled under the

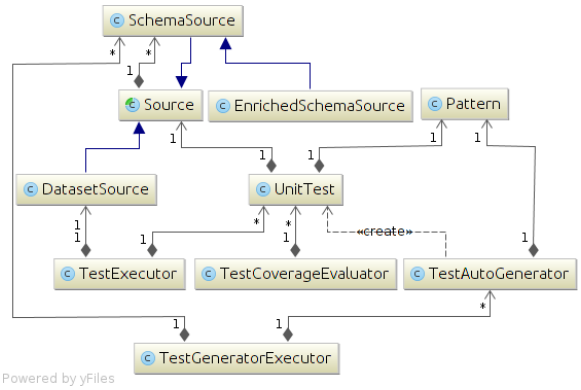

Figure 3: Then main components of the core Databugger library as a UML diagram. The diagram was generated with the Intellij IDEA program.

Databugger ontology[8]. The input and output of the tool is entirely in RDF which makes it highly configurable. One can read the input (patterns, TAGs and test cases) from the file system as RDF files, dereference then from a remote location or retrieve them from a SPARQL endpoint. Concrete test cases are equipped with persistent identifiers to facilitate test tracking over time. Our pattern library uses SPARQL1.1 and *property paths*[9] for properly checking transitive violations (e.g. `rdfs:domain` and `rdfs:range`).

Databugger was built with *Java* and the *Jena* framework. A UML diagram of the core library is depicted in Figure 3. The main components of the Databugger Library are the *Source*, *Pattern*, *TestAutoGenerator* and *UnitTest*. A *Source* represents an arbitrary RDF source uniquely identified by a URI. Concrete *Source* implementations are:

[3] http://databugger.aksw.org

[4] A screencast of the tool is available at http://youtu.be/3g9R3P1kwdw

[5] http://databugger.aksw.org/data/patterns#

[6] http://databugger.aksw.org/data/tests#

[7] http://databugger.aksw.org/data/generators#

[8] http://databugger.aksw.org/ns/core#

[9] http://www.w3.org/TR/sparql11-property-paths/

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 1: 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 as the total number of errors that occurred from manual (ManEr) and enriched (EnrEr) tests. The last column shows the average errors per distinct subject.

- *SchemaSource*: a schema, a vocabulary or an ontology in RDF, e.g. `skos`. The framework can automatically dereference a schema from a URI, a file location or a prefix (e.g. `foaf`). For prefix resolution we query the LOV SPARQL endpoint[10] to get a dereferenceable URI. Using LOV we provide easy test access to commonly used vocabularies.
- *EnrichedSchemaSource*: a semi-automatically enriched schema as explained in Section 2.
- *DatasetSource*: an RDF dataset accessible via a SPARQL endpoint, e.g. `DBpedia`[11]. A subset of the dataset can be selected by providing a list of Named Graphs. RDF dump datasets will be supported in the following releases of Databugger.

The *Pattern* component holds a DQTP and *UnitTests* are instantiations of *Patterns* generated by binding a pattern placeholder to valid replacements. According to our methodology, a *UnitTest* can be created either manually or automatically. The automatic *UnitTest* generation is performed by the *TestAutoGenerator* (TAG) component. Each TAG is based on a *Pattern* and automatically instantiates test cases (*UnitTests*) for an input schema.

The *TestGeneratorExecutor* component takes as input a dataset and a list of schemas, for each schema a) automatically generating test cases and b) loading any existing manually defined test cases. Additionally, any pre-defined test cases for the dataset are loaded. The automatically generated test cases are cached locally for future reference. The output of this component is passed to the *TestExecutor* component which executes the test cases against a *DatasetSource*. The *TestCoverageEvaluator* is an optional step that calculates the test coverage of a test case set against a dataset. However, this step requires precalculated property and class statistics of the dataset. Future versions of the library will be able to autogenerate these statistics.

4. TEST RESULTS

To showcase the re-usability of our automatically and manually generated test cases, Databugger was run against the following five datasets for evaluation: The English[12] and Dutch[13] DBpedia, LinkedGeoData[14], id.loc.gov[15] and datos.bne.es[16]. An overview of the dataset assessment is provided in Table 1. A detailed description of the evaluation results can be found in [5]. Additionally, by running the *TestGeneratorExecutor* component against all the registered Linked Open Vocabularies (LOV) we managed to generate a total of 32,293 unique reusable test cases for 297 common vocabularies.[17]

5. CONCLUSIONS AND FUTURE WORK

In this paper, we described *Databugger*, a framework for test-driven Linked Data quality assessment. We introduced the methodology behind Databugger and described in detail the architecture, usability and extensibility of the tool. Additionally we provided an evaluation overview on five different datasets and a library of 32,293 unique test cases for 297 common vocabularies. In future versions of the tool, we plan to further enrich our pattern library and increase the *Test Auto Generators* for full `RDFS` and `OWL` coverage. Additionally we plan to support testing of arbitrary RDF dumps and fully automated test-coverage calculation. Finally we will investigate the need of a RESTfull testing service as well as a testing server for streaming RDF sources.

6. REFERENCES

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

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

[3] 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.

[4] J. M. Juran. *Quality Control Handbook*. McGraw-Hill, 4th edition, August 1988.

[5] D. Kontokostas, P. Westphal, S. Auer, S. Hellmann, J. Lehmann, and R. Cornelissen. Test-driven evaluation of linked data quality. In *WWW*, 2014 (to appear).

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

[11] `http://dbpedia.org`

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

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

[17] `https://github.com/AKSW/Databugger/tree/master/archive/WWW_2014`