Modelling and Analysing Dynamic Linked Data using RDF and SPARQL

Tobias Käfer, Alexandra Wins, and Maribel Acosta

Institute AIFB, Karlsruhe Institute of Technology (KIT), Germany
{tobias.kaefer|maribel.acosta}@kit.edu
alexandra.wins@student.kit.edu

Abstract. Analyses of dynamic Linked Data are inherently dependent on changes in RDF graphs (logical level) and what happens on the HTTP and networking level (physical level). However, these dependencies have been reflected in previous works only to a limited extent, which may lead to inaccurate conclusions about the dynamics of the data. To overcome this limitation, we tackle the problem of modelling dynamic Linked Data to capture changes both at the logical and physical level of Linked Data. We present our work in progress in this paper. We propose an RDF model of descriptions of both the HTTP requests/responses/networking errors made when downloading, and the RDF data thus obtained. The model allows for carrying out more comprehensive analyses of dynamic Linked Data in a declarative fashion. We present a processing pipeline to distil such modelled RDF data from datasets created using LDspider, such as the Dynamic Linked Data Observatory. We show the usefulness of our model by repeating three analyses of a previous paper on the Dynamic Linked Data Observatory using SPARQL queries, which we executed on data that follows our proposed model on three SPARQL engines.

1 Introduction

Linked Data is dynamic: we see data in Linked Data documents being updated, or documents, or entire servers, go on- or offline. This dynamics has been acknowledged and analysed on different levels of abstraction, ranging from data access and data syntax [15] to data schema [8]. Yet, the interdependence of the dynamics that can be observed on the different levels of abstraction is still uncharted territory. For instance, previous analyses on higher level of abstraction did not reflect the dynamics caused by data access [8, 11, 7, 17]. We argue that analyses of this interdependence have been neglected because there is no uniform way to access all the required information, and the necessity of the analyses has been overlooked as previous analyses used code in imperative languages, where the assumptions are hidden in the control flow of the code. For instance:

Motivating Example. In a previous paper, the number of overall documents appearing in the Dynamic Linked Data Observatory dataset, which monitors a set of 95'737 seed URIs, was reported to be 86'696 [15]. To be precise, this was the number of information resources ever encountered in any of the 29 snapshots

available at that time and determined by counting the contexts of quads. When considering e. g. 171 snapshots, this number rises to about 106'105 documents, more than seed URIs. This mismatch points us towards that we may have to re-ask our question and take into account the dereferencing process: If we instead consider the number of seed URIs that dereferenced to information resources, the number for 29 snapshots is 92'712. To further put this number into context, we note that for the first snapshot, 67'107 seed URIs were redirected, ie. had non-trivial dereferencing, and 77'958 could be dereferenced successfully.

We want to lower the barrier of entry to analyses that take the interdependence between data and data access into account by tackling the problem of modelling dynamic Linked Data in a way that takes into account both, the data access and the data itself. Moreover, we want to make the analyses more easy to validate by using a declarative approach for the analyses.

Data following the Linked Data principles has been published at a vast scale. Such data is meant to reflect the state of things in the world, which is without a doubt dynamic. Moreover, with the advent of Read-Write Linked Data –where Linked Data is meant to be changed using HTTP requests– and the Web of Things –where Linked Data provides access to ever-changing sensor values– we expect a further increase of dynamic Linked Data.

Knowledge about the dynamics of Linked Data is relevant e. g. in choosing architectures for Linked Data applications (ranging from data warehousing to live querying) and optimising Linked Data processing systems (e. g. indexing strategies) [19]. To investigate the dynamics of Linked Data, Käfer et al. set up the Dynamic Linked Data Observatory [16]. This observatory creates weekly snapshots of data and meta data from RDF documents retrieved from the web. Several works have investigated the dynamics of Linked Data with analyses written in imperative languages that post-process the snapshots collected by the observatory [8, 7, 11]. Such imperatively given analyses may intransparently introduce assumptions, which hinder the reproducibility and the validity of the results. In contrast, declaratively modelled analyses raise the understandability of the analysis process and facilitate the validation of the analyses and the results.

In this paper, we present ongoing work to model dynamic Linked Data using RDF such that such interdependencies can be investigated more easily. The proposed model enables declarative analyses of dynamic Linked Data using the SPARQL query language. In summary, the main contributions of our work are as follows:

- A data model that relies on RDF and captures the physical and logical aspects of dynamic Linked Data
- A processing pipeline that exploits the information recorded in crawler logs to generate relevant meta data about dynamic Linked Data
- A proof concept of our proposed approach that includes: five snapshots of Linked Data represented with the proposed data model, and three SPARQL queries to analyse the dynamics of Linked Data

The paper is structured as follows: In Section 2, we introduce the standards and practices around Linked Data, as relevant for our analyses. Moreover, we

hint at the technical details of the crawler employed, which shape our proof of concept. In Section 3, we present the data model and a processing pipeline to derive data according to the model. Subsequently, in Section 4, we show the applicability of our approach by giving queries and results as proof of concept. Based on our proof of concept, we discuss future refinements of the approach in Section 5. Next, in Section 6, we present related work. Last, in Section 7, we summarise and discuss our approach, and outline future directions for our work.

2 Preliminaries

In this section, we describe Linked Data with special focus on the features we need for generating the RDF description of the time series that incorporates both request meta data and the RDF payload from Linked Data. Moreover, we describe the Dynamic Linked Data Observatory, a project that publishes a time series of RDF data and log files. For the description of our extraction, we give the necessary introduction to LDSpider, the Linked Data crawling software used in the Dynamic Linked Data Observatory, particularly its output data.

Linked Data Linked Data is a set of principles for publishing data on the web [3]. The principles advocate the use of web standards for data on the web: URIs as identifiers, HTTP-GET for data access, and RDF as data model.

URI Uniform Resource Identifiers (URIs) are names for *things* on the web in the form of character sequences [4]. URIs start with a scheme, where the scheme `http` denotes that HTTP-based communication with the resource may be possible. In addition, URIs may refer to: (i) *non-information resources* that denote abstract or physical things, e. g. a URI of the moon is `http://dbpedia.org/resource/Moon`; (ii) *information resources* that denote the RDF documents that describe the things, e. g. the URI of a document that describes the moon is `http://dbpedia.org/data/Moon.n3`.

HTTP The Hypertext Transfer Protocol (HTTP) is a protocol for data transfer on the web [9]. Data exchange is subdivided into requests and responses. Requests are sent to URIs, which return a response. Requests do not return in the case of networking issues and server outages, which are outside of the HTTP framework. There are a number of request methods, from which Linked Data uses the GET method for retrieving state. To send a GET request to a URI is also called *dereferencing* the URI. A HTTP response consists in (1) a status line reporting about the status of the request using a numeric status code and a textual explanation, e. g. in the case of a successful request the status line reads `200 OK`, (2) optional headers with meta data about the response, and (3) an optional body, which contains RDF data in the case of Linked Data. Status codes are three-digit integers, where the first digit determines the status code class. The HTTP specification distinguishes the following classes [10]:

- 1xx (Informational): The request was received, processing continues
- 2xx (Successful): The request was received and can be successfully answered
- 3xx (Redirection): The request needs further client action for completion
- 4xx (Client Error): The request cannot be fulfilled due to a client error
- 5xx (Server Error): The request cannot be fulfilled due to a server error

The responses with 3xx status code typically contain a `Location` header with a URI to which the server redirects the client for the next request. On the Linked Data web, the use of redirects is made to distinguish two classes of URIs: Those referring to information resources and those referring to non-information resources. Information resources are differentiated when their URI is dereferenced, return a successful response with non-empty body (e. g. a RDF document about the moon). Non-information resources are all other resources (e. g. the moon). Previous analyses focussed on the mere data from the response bodies of successful requests (after following redirects if applicable). With the modelling presented in this paper, we want to enable analyses that also take into account unsuccessful HTTP requests and the process of following redirects when dereferencing.

RDF, Triples, Quads The Resource Description Framework (RDF) is a graph-based data model [6] used in Linked Data. An RDF graph is a directed graph, where labelled nodes are connected by labelled arcs. An RDF graph G is composed of a set of triples, where a triple $t = (subject,\ predicate,\ object)$ such that $t \in (\mathcal{U} \cup \mathcal{B}) \times (\mathcal{U}) \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$, where $\mathcal{U}$ denotes the set of all URIs, $\mathcal{B}$ the set of all blank nodes, and $\mathcal{L}$ the set of all literals. A triple can be extended with a fourth element called "context" to form a quad. In this paper, the term quad refers to a triple plus the URI of the information resource where the triple has been obtained from, as described in [20, § 3.5].

The Dynamic Linked Data Observatory The Dynamic Linked Data Observatory is a dataset containing a time series of Linked Data from the Web [16]. The time series is being collected weekly since May 2012. For the composition of the time series, a seed list of 95,737 URIs is dereferenced each week (following redirects) and both the retrieved RDF data and meta data in the form of log files is collected. While the Dynamic Linked Data Observatory also crawls from the seed list, we neglect the crawling part in this paper. The data collection in the Dynamic Linked Data Observatory is done using LDSpider. We use the data from the Dynamic Linked Data Observatory to show the applicability for the modelling/querying approach we propose in this work.

LDSpider LDSpider [14] is a crawler for Linked Data. In the Dynamic Linked Data Observatory, LDSpider is used to download the seed list following links and crawl from the seed list. LDSpider produces five types of outputs. The first output of LDSpider is (1) `Data` that contains quads data obtained from dereferencing the seed URIs. The quads contain the triples from the download, and in context position the URI of the information resource from which the triple

was downloaded. Note that the context URI is not always the URI from the seed list: If the request to the URI from the seed list was answered using a redirect to another URI, LDSpider dereferences that URI also. If the GET request to the latter URI was answered successfully with RDF data, this is the data and the context URI that go into the quad. Other relevant output of LDSpider is an (2) `HTTP Status` log with information about the status codes in the responses to the HTTP requests the crawler performed. Moreover, LDSpider outputs (3) `Redirects` that records pairs describing from which URI to which URI the crawler has been redirected. In addition, LDSpider records the (4) `HTTP Headers` from the HTTP responses. While this meta data is quite comprehensive when it comes to information on the application layer (i. e. the responses received), is not sufficient to describe the results of all HTTP requests that have been performed. Lastly, there is the (5) `Standard Error` output of the crawler containing e. g. networking exceptions encountered during crawling. For the destillation of data from the Dynamic Linked Data Observatory according to the model we propose in this work, we have to consider the different outputs of LDSpider.

3 Our Approach

In our proposed solution, first we provide a model based on RDF to represent information both at the physical and logical levels of Dynamic Linked Data. Then, we propose a processing pipeline to derive the RDF data following our model from current Linked Data monitoring approaches.

3.1 Modelling Dynamic Linked Data in RDF

Typically, to capture the dynamics of Linked Data, monitoring approaches periodically dereference snapshots of data from a set of seed URIs. Therefore, we propose a model to represent snapshots of Linked Data by capturing the entire dereferencing process of data from a list of seed URIs. The proposed model then allows for analysing changes among time series of Linked Data by comparing the data recorded in each snapshot. The proposed model is depicted in Figure 1. We use the UML class diagram with the following correspondence from UML to RDFS: UML classes depict `rdfs:Class`es and UML associations depict `rdfs:domain` and `rdfs:range` of an `rdf:Property`. We use `list:member`[1] associations to state the `rdfs:Class` of the members of an `rdf:List`.

In our model, every Linked Data snapshot is annotated with a timestamp. The core concept in each snapshot is an observation associated with a seed URI.

To represent the details about the **physical level** of Dynamic Linked Data, our model captures in an RDF list the HTTP requests following redirects where dereferencing a seed URI. In this way, the model preserves the order in which the requests have been made. RDF lists are closed and ordered, both features are desired for the requests, as the number of requests that has been made is

[1] The prefix `list` meaning `http://www.w3.org/2000/10/swap/list#`

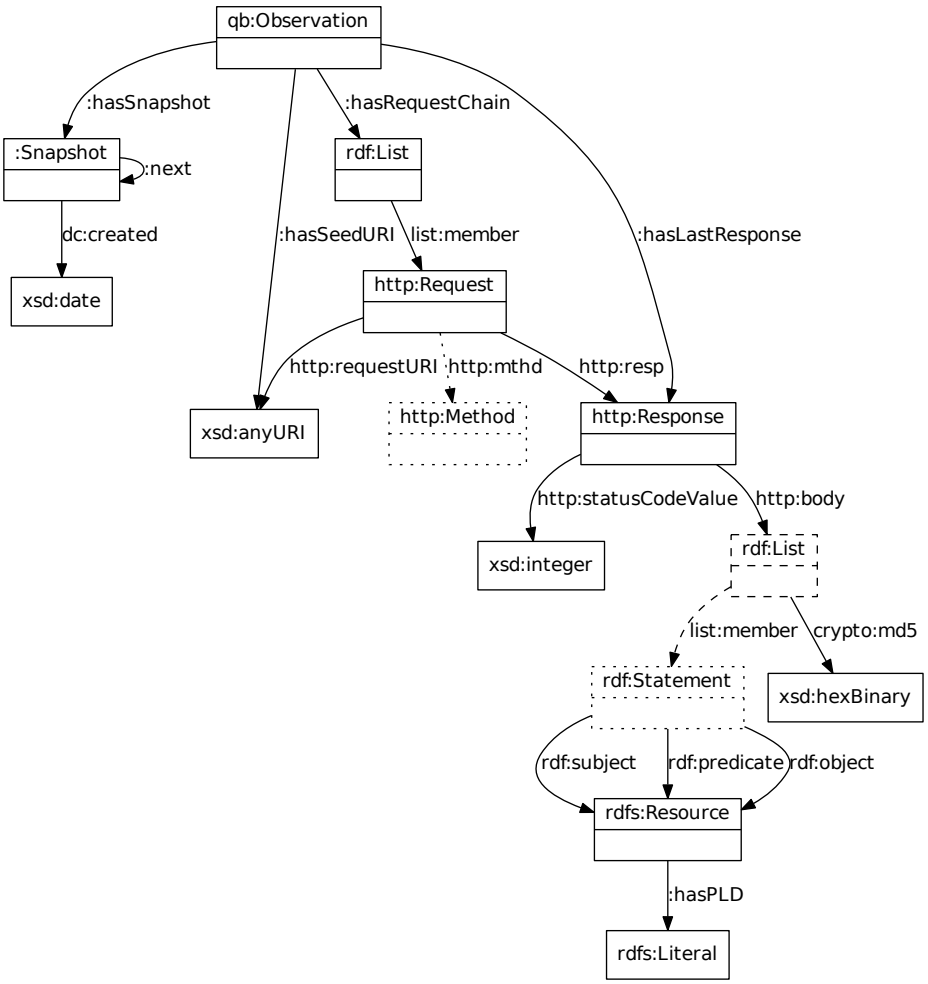

Fig. 1. The proposed model to represent Dynamic Linked Data. Each snapshot of Linked Data is annotated with a timestamp. Each observation records the process of dereferencing a seed URI. The RDF graphs obtained from dereferencing each seed URI is represented as a list of reified statements. The dashed and dotted elements in the model are possible places for improvement.

determined by what happened, and the order of the request is determined by the redirects. Nonetheless, it is important to highlight that to traverse the elements in RDF lists in queries it is necessary to use SPARQL property paths, which may be rather expensive to evaluate depending on the assumed semantics [1]. To allow for querying the information resource of a seed URI, the model includes the `:hasLastResponse` property. In this way, it is not necessary to traverse the entire list of requests thus reducing the complexity of queries that only require data

about the information resource associated with a seed URI. For each request, the model represents the obtained response which contains the status code and the response body (if applicable).

The **logical level** of Dynamic Linked Data corresponds to the RDF graphs obtained in the body of a response. To associate these graphs with the corresponding response, the proposed model represent the triples in the RDF graphs as an RDF list of reified statements. In this context, reified statements allows for referring to RDF graphs at different points in time. Another option would have been named graphs for each information resource at each point in time. Nonetheless, this option has several drawbacks. First, we would have to name each named graph in `FROM` clauses of analysis queries. With over 210 snapshots of about 95'737 URIs, we would have more than 20M `FROM` clauses in the queries, which is rather lengthy. Second, to try different triple stores, whose treatment of named graph varies if they are supported (some treat the triples in the named graphs as asserted in the default graph), we want to rely on common features independently from the triple store. In addition to the reified statements, the model contains a hash of the full graph as binary using the `crypto:md5` property[2]. This allows for efficiently detecting RDF graphs that change over time.

3.2 Processing Pipeline to Extract the Dynamics of Linked Data

The processing pipeline includes the handling of the data produced by the monitoring Linked Data tools. In our approach, the pipeline obtains relevant information about the dynamics of the crawled Linked Data that is scattered in different LDSpider outputs into one RDF graph.

An overview of the processing pipeline is presented in Figure 2. The proposed pipeline includes two components: one to process the physical level of Linked Data (meta data)[3], and another to process the logical level of Linked Data (RDF data)[4]. It is important to highlight that the pipeline generates URIs for the requests to the information resources (which is the only information that appears in both, data and meta data) such that the output of the two components can be correctly integrated using RDF merge. The merged data then follows the model presented in Figure 1. The meta data processing code adds custom HTTP status codes for URIs whose dereferencing yields non-HTTP errors such as networking errors. Those custom HTTP status codes allow us to query both HTTP errors and networking errors in a uniform fashion.

An important aspect when processing the logical level of Linked Data is the correct handling of blank nodes. In the presence of blank nodes, the same RDF graph may use different blank node identifiers to represent the same data. Hence, in our approach we detect isomorphisms among RDF graphs from different snapshots and replace the blank nodes by URIs using the hash-based skolemisation approach described by Hogan in [13]. This allows for checking whether a triple

[2] With the prefix `crypto` meaning `http://www.w3.org/2000/10/swap/crypto#`
[3] `http://github.com/kaefer3000/dyldo-http2rdf`
[4] `http://github.com/kaefer3000/dyldo2qb`

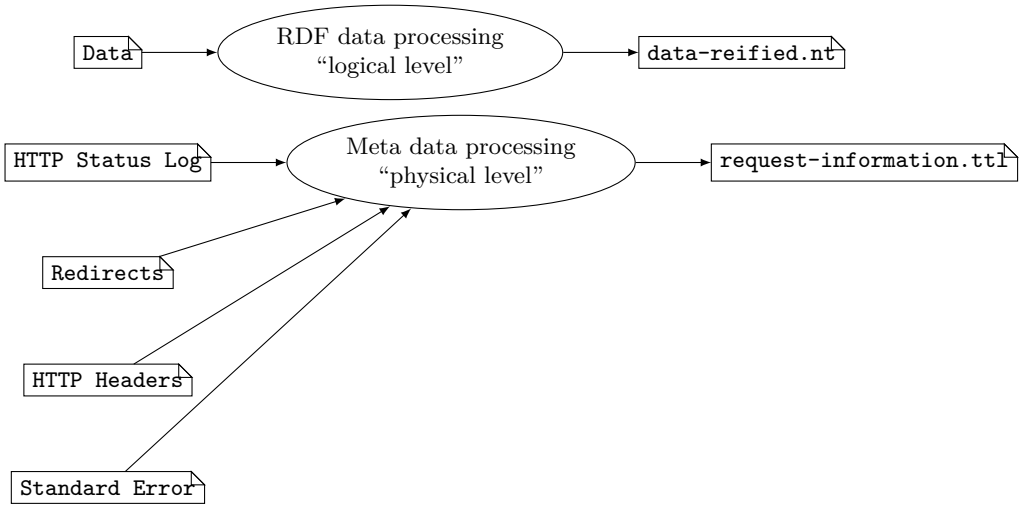

Fig. 2. The proposed processing pipeline. The input (on the left) is produced by LD-Spider, the output (on the right) is RDF data using the proposed model in Figure 1.

with a blank node from the graph derived at one point in time is the same as a triple with a blank node from another point in time using RDF URI equality.

4 Applicability of Our Approach

In this section, we repeat three analyses from the 2013 paper of Käfer et al. [15] using SPARQL queries on the data distilled from the first five snapshots of the Dynamic Linked Data Observatory. We first present the queries, numbered according to the number of their corresponding figure in [15]. Then, we report on the run time of the query execution on different SPARQL engines.

4.1 Queries

Q1 The first analysis, depicted in Figure 1 of [15] dubbed “appearances of documents”, asked how many documents appeared in which number of snapshots. The aim of the analyses was to investigate the availability of Linked Data documents. In the spirit of our motivating example, we changed the analysis to the share of seed URIs that dereferenced successfully. In the query, the commented line additionally checks for non-empty HTTP response bodies. We give a query for this analysis in Figure 3. We used the query results to produce Figure 4.
Q2 While the first analysis only looked at successful dereferencing, the second analysis (Figure 2 of [15]) looked more closely at the availability by investigating all HTTP status codes returned in the dereferencing process. We give a query for that analysis in Figure 5. The corresponding produced visualisation can be found in Figure 6.

```
PREFIX : <http://purl.org/dyldo/vocab#>
PREFIX http: <http://www.w3.org/2011/http#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT (COUNT(?seedURI) AS ?uris) ?NoSnapshots WHERE {
  {
  SELECT (COUNT(?snapshot1) AS ?NoSnapshots) ?seedURI  WHERE {
      ?observation :hasSnapshot ?snapshot1;
                     :hasSeedURI ?seedURI;
                    :hasLastResponse ?res .
      ?res http:statusCodeValue "200"^^xsd:integer .
      # ?res http:body ?body.
    } GROUP BY ?seedURI
  }
} GROUP BY ?NoSnapshots
```

Fig. 3. Query Q1 to investigate the appearance of documents. Cf. Figure 1 of [15]. The commented line would check for an non-empty HTTP body.

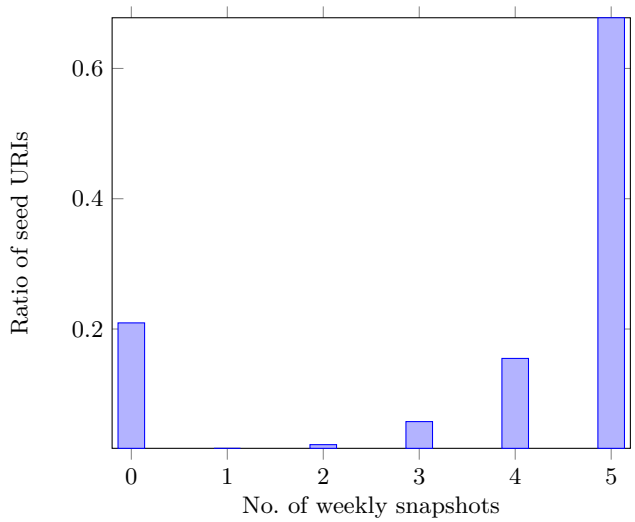

Fig. 4. The appearance of documents. Cf. Figure 1 from [15], created using the results from Q1 in Figure 3.

```
PREFIX : <http://purl.org/dyldo/vocab#>
PREFIX dc: <http://purl.org/dc/terms/>
PREFIX http: <http://www.w3.org/2011/http#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?snapshotDate
       (CONCAT(?scDigit1, "xx") AS ?statusClass)
       (COUNT(DISTINCT ?seedURI) AS ?seedUriCount)
WHERE {
  ?observation :hasSnapshot ?snapshot ;
    :hasSeedURI ?seedURI ;
    :hasLastResponse ?res .
  ?res http:statusCodeValue ?sc .
  BIND(SUBSTR(STR(?sc), 1, 1) AS ?scDigit1)
} GROUP BY ?scDigit1 ?snapshotDate
```

Fig. 5. Query Q2 to investigate the distribution of HTTP response classes, cf. Figure 2 from [15].

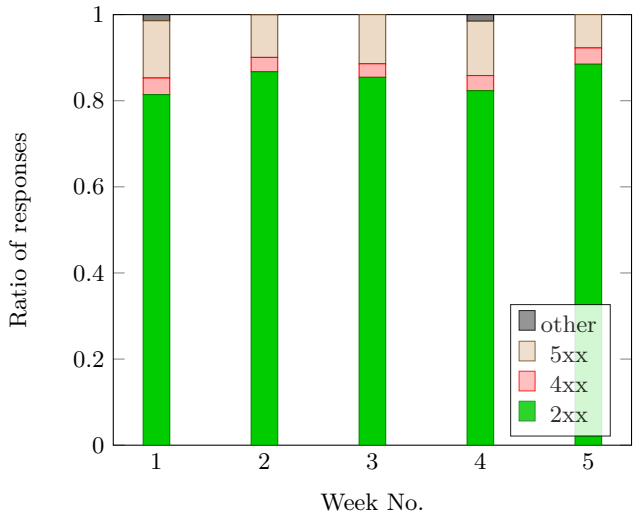

Fig. 6. The distribution of HTTP response codes encountered when dereferencing the seed URIs, cf. Figure 2 from [15], created using the results from Q2 in Figure 5.

Q5 To evaluate more challenging SPARQL queries, we last have a look at Figure 5 of [15]. This figure is the first in [15] to investigate change in the Linked Data sources, i. e. it involves multiple snapshots in the query. The query looks at which number of URIs had which number of changes.

4.2 Data Loading and Query Execution

To showcase the applicability of our approach, we present loading and querying times for the first five snapshots and the three queries.

From the five snapshots, we yielded about 10M triples overall with information about the physical level (including the hashes on the graphs) and 481M triples overall with the reified data (derived from about 80M triples raw data), the hashes, and data about pay-level domains required for more analyses from [15]. As the snapshots are processed individually and because we exploit in the model the fact that many sources do not change so much over time, there is a high number of duplicate triples between the data from different snapshots. Therefore, the data to be processed by the SPARQL query engines is considerably lower than the reported triple numbers.

We ran the experiments on a Debian 8 (jessie) 64 bit Linux system with 4 cores of an AMD Opteron 62xx with 2 GHz and 48 GB of RAM. We report the times in Table 1. We report the loading times for the physical data (including the hashes on the graphs), and the full data. Moreover, we briefly report on the lessons learned while using different SPARQL query engines to evaluate our approach:

Virtuoso[5] (Version 7.20.3217) was not able to load the full data due to the skews in the data introduced by the reified triples. We nevertheless include Virtuoso in our measurements, as we may switch the representation of the logical data in the future.

Blazegraph[6] (Version 2.1.4) managed to load the entire data, but query performance is low when using the automatic query optimiser for queries where different snapshots are compared, for instance it took about a week to get results for Q5. Optimising the query plan using Blazegraph's hints allowed us to significantly improve the performance, for Q5 down to 73 s.

Linked Data-Fu[7] (Version 0.9.12) [18] does not index the data but queries RDF on the fly, so we there are no loading times. Moreover, Linked Data-Fu does not support aggregates, so we had to remove them from the queries and implemented them using AWK[8] scripts. For the query evaluation, we piped the results from Linked Data-Fu through AWK 4.1.1 to get the envisioned results. We report the overall run time in the table for processing the physical information and hashes.

[5] `http://virtuoso.openlinksw.com/`
[6] `http://www.blazegraph.com/`
[7] `http://linked-data-fu.github.io/`
[8] `http://www.gnu.org/software/gawk/`

```
PREFIX : <http://purl.org/dyldo/vocab#>
PREFIX crypto: <http://www.w3.org/2000/10/swap/crypto#>
PREFIX http: <http://www.w3.org/2011/http#>
PREFIX qb: <http://purl.org/linked-data/cube#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?changeCount (COUNT (?changeCount) as ?uriCount) WHERE {
  SELECT (COUNT(?seedURI) AS ?changeCount) WHERE {
    ?observation1 a qb:Observation ;
      :hasSnapshot ?snapshot;
      :hasLastResponse ?resp1 .
    ?resp1 http:statusCodeValue "200"^^xsd:integer ;
      http:body [ crypto:md5 ?hash1 ] .
    ?observation2 a qb:Observation ;
      :hasSnapshot ?snapshot2 ;
      :hasLastResponse ?resp2 .
    ?resp2 http:statusCodeValue "200"^^xsd:integer ;
      http:body [ crypto:md5 ?hash2 ] .
    ?snapshot :next ?snapshot2 .
    FILTER((STR(?hash1) != STR(?hash2)))
  } GROUP BY ?seedURI
} GROUP BY ?changeCount
```

Fig. 7. Query Q5 to investigate the number of changes per number of URIs, cf. Figure 5 from [15].

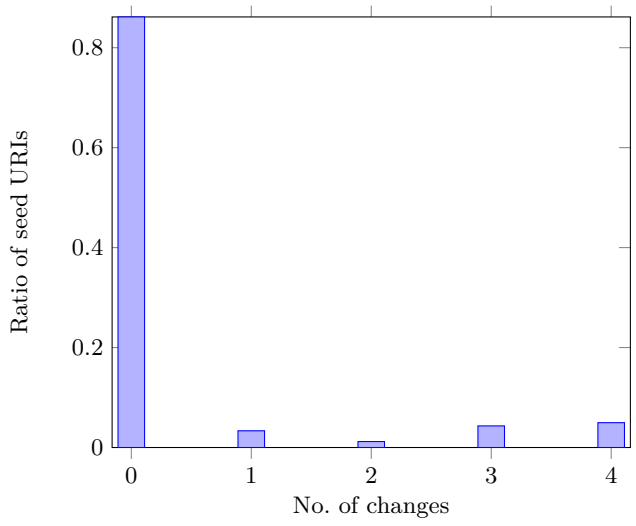

Fig. 8. Number of changes per number of URIs. Cf. Figure 5 from [15], created using the results from Q5 in Figure 7.

Table 1. Times for loading and querying data.

	Loading		**Querying**		
	physical	full	Q1	Q2	Q5
Virtuoso 7.20.3217	181 s	failed	4 s	18 s	14 s
Blazegraph 2.1.4	289 s	6 h	45 s	80 s	73 s*
Linked Data-Fu 0.9.12 + AWK 4.1.1	n/a	n/a	97 s	112 s	178 s

∗ Using optimisation hint. Otherwise, the execution time was about 1 week.

5 Discussion of the Approach

We observed that our particular workload poses challenges to indexes and query optimisers. Optimisation of the queries by re-ordering or re-formulating parts of the query can take the processing time from days down to minutes. Another line of optimisation is the data: For instance, we could reduce the overall data by omitting triples that are depicted dotted in Figure 1 because they are the same for all requests or triples. Second, we could use less verbose modelling for the triples that are depicted dashed in the figure: We use the RDF list to describe the HTTP body because it is terminated. The order of the statements does not matter. As we want to use SPARQL for querying, where the closed-world assumption is made, we can reduce the number of triples by introducing a blank node or URI for the HTTP response body, and use triples with a predicate like `rdfs:member` to connect the body to the statements in the RDF graph of the response body.

6 Related Work

In this section, we analyse current approaches for monitoring Linked Data.

The Dynamic Linked Data Observatory [16] is a framework that monitors the dynamics of Linked Data. The Dynamic Linked Data Observatory crawls RDF documents available on the web periodically and provisions logs and raw data about the crawling process. The data generated by the the Dynamic Linked Data Observatory is key for tracking changes in Linked Data sets, however, it requires further processing in order to extract relevant information about the dynamics of Linked Data sets. Therefore, in this work we propose an approach that enriches the data generated by the Dynamic Linked Data Observatory. In its raw form, the data from the Dynamic Linked Data Observatory has been analysed by different scholars, mostly without taking networking aspects into account [8, 11, 7, 17, 15].

Other approaches have focused on monitoring other aspects of Linked Data [2, 12, 5]. For example, LODStats [2] is an approach that collects statistics about RDF datasets available on the web. LODStats provides declarative descriptions of datasets using the LODStats Dataset Vocabulary (LDSO). LDSO extends the

Vocabulary of Interlinked Datasets[9] (VoID) and the Data Catalog Vocabulary[10] (DCAT) to model meta data and statistical metrics about Linked Data sets. Furthermore, the work by Hasnain et al. [12] focuses on providing a catalogue of SPARQL queries to compute statistics based on VoID descriptions of RDF datasets available through SPARQL endpoints. SPARQLES [5], in turn, focuses on monitoring publicly available SPARQL endpoints. SPARQLES provides a set of predefined queries to inspect the support of SPARQL features and performance of endpoints. In contrast to our proposed approach, related works focus on reporting statistics about the current state of the datasets.

7 Conclusion and Future Work

In this paper, we presented an RDF model of dynamic Linked Data for declaratively analysing dynamic Linked Data time series using SPARQL queries. We provide an implementation to distil data according to the model from data collected using LDSpider, such as the Dynamic Linked Data Observatory. We gave three SPARQL queries for analysing dynamic Linked Data and evaluated them over five snapshots from the Dynamic Linked Data Observatory using three different SPARQL engines.

For future work, we want to run more queries on more snapshots. As the data that is emitted from our processing code does not have many selective predicates, the engineering of data and queries seems not to be a trivial undertaking.

Acknowledgements

This work is partially supported by the German federal ministry of education and research in AFAP, a Software Campus project (FKZ 01IS12051).

Bibliography

1. Arenas, M., Conca, S., and Pérez, J.: Counting beyond a Yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard. In: Proceedings of the 21st International Conference on World Wide Web (WWW) (2012)
2. Auer, S., Demter, J., Martin, M., and Lehmann, J.: LODStats - An Extensible Framework for High-Performance Dataset Analytics. In: Proceedings of the 18th International Conference on Knowledge Engineering and Knowledge Management (EKAW) (2012)
3. Berners-Lee, T.: Linked Data. Design Issues, (2006). `http://www.w3.org/DesignIssues/LinkedData.html`
4. Berners-Lee, T., Fielding, R., and Masinter, L.: *Uniform Resource Identifier (URI): Generic Syntax*. Internet Standard. RFC 3986. IETF (2005).
5. Buil Aranda, C., Hogan, A., Umbrich, J., and Vandenbussche, P.: SPARQL Web-Querying Infrastructure: Ready for Action? In: Proceedings of the 12th International Semantic Web Conference (ISWC) (2013)

[9] `http://www.w3.org/TR/void/`
[10] `http://www.w3.org/TR/vocab-dcat/`

6. Cyganiak, R., Wood, D., and Lanthaler, M., eds.: RDF 1.1 Concepts and Abstract Syntax. Recommendation, W3C. (2014). http://www.w3.org/TR/rdf11-concepts/
7. Dividino, R. Q., Gottron, T., and Scherp, A.: Strategies for Efficiently Keeping Local Linked Open Data Caches Up-To-Date. In: Proceedings of the 14th International Semantic Web Conference, (ISWC) (2015)
8. Dividino, R. Q., Scherp, A., Gröner, G., and Grotton, T.: Change-a-LOD: Does the Schema on the Linked Data Cloud Change or Not? In: Proceedings of the Fourth International Workshop on Consuming Linked Data (COLD) at the 12th International Semantic Web Conference (ISWC) (2013)
9. Fielding, R. and Reschke, J., eds.: *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. RFC 7230 (Proposed Standard). IETF (2014).
10. Fielding, R. and Reschke, J.: *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. RFC 7231 (Proposed Standard). IETF (2014).
11. Gottron, T. and Gottron, C.: Perplexity of Index Models over Evolving Linked Data. In: Proceedings of the 11th European Semantic Web Conference (ESWC) (2014)
12. Hasnain, A., Mehmood, Q., Sana e Zainab, S., and Hogan, A.: SPORTAL: Profiling the Content of Public SPARQL Endpoints. International Journal on Semantic Web and Information Systems 12(3) (2016)
13. Hogan, A.: Skolemising Blank Nodes while Preserving Isomorphism. In: Proceedings of the 24th International Conference on World Wide Web (WWW) (2015)
14. Isele, R., Umbrich, J., Bizer, C., and Harth, A.: LDSpider: An open-source crawling framework for the Web of Linked Data. In: Proceedings of Posters and Demos at the 9th International Semantic Web Conference (ISWC) (2010)
15. Käfer, T., Abdelrahman, A., Umbrich, J., O'Byrne, P., and Hogan, A.: Observing Linked Data Dynamics. In: Proceedings of the 10th European Semantic Web Conference (ESWC) (2013)
16. Käfer, T., Umbrich, J., Hogan, A., and Polleres, A.: Towards a Dynamic Linked Data Observatory. In: Proceedings of the 5th Workshop on Linked Data on the Web (LDOW) at the 25th International Conference on World Wide Web (WWW) (2012)
17. Nishioka, C. and Scherp, A.: Information-theoretic Analysis of Entity Dynamics on the Linked Open Data Cloud. In: Proceedings of the 3rd International Workshop on Dataset PROFIling and fEderated Search for Linked Data (PROFILES) at the the 13th European Semantic Web Conference (ESWC) (2016)
18. Stadtmüller, S., Speiser, S., Harth, A., and Studer, R.: Data-Fu: A Language and an Interpreter for Interaction with Read/Write Linked Data. In: Proceedings of the 22nd International Conference on World Wide Web (WWW) (2013)
19. Umbrich, J., Karnstedt, M., Hogan, A., and Parreira, J. X.: Hybrid SPARQL Queries: Fresh vs. Fast Results. In: Proceedings of the 11th International Semantic Web Conference (ISWC) (2012)
20. Zimmermann, A., ed.: RDF 1.1: On Semantics of RDF Datasets. Working Group Note, W3C. (2014). http://www.w3.org/TR/rdf11-datasets/