Enabling the Scholarly Discourse of the Future: Versioning RDF Data in the Digital Humanities

Martina Bürgermeister
University of Graz
Graz, Austria

Abstract

The dynamic and collaborative scholarly landscape of the humanities and cultural sciences is in urgent need of reliable knowledge about the origin and genesis of its data. Versioning, with its capability to document data and its modifications, can be used to integrate and provide this critical information. In the following paper, the versioning of RDF data is presented as a method to make research more trustworthy, and as a means of turning the processes through which data is changed over time into objects of study within the digital humanities.

1 Introduction

Scholarly work and the knowledge that it generates are the result of iterative processes. Making these processes transparent is an indispensable part of scholarly communication: it serves as the basis for trust in data, which in turn encourages the use of that data in subsequent research (Borgman, 2010; Zimmerman, 2008).

The social need for trust in data is reflected in the concept of the Semantic Web, where the legitimacy of data plays a fundamental role. There are several ways to render data more trustworthy, one of them being the provision

In: Tara Andrews, Franziska Diehr, Thomas Efer, Andreas Kuczera and Joris van Zundert (eds.): Graph Technologies in the Humanities - Proceedings 2020, published at http://ceur-ws.org.

of metadata concerning the origin of data and the modifications it has undergone over time:

> [N]ot only could such provenance metadata be used as selection criteria, their existence may encourage scholars to contribute data to the community, as credit could be assigned more accurately and publicly. Provenance metadata also could enable processed data to be 'rolled back' through transformations for the purposes of correcting errors at interim stages or computing different transformations (Borgman, 2010, pp. 131-132).

Documenting data and its modifications and, if necessary, reversing those modifications, is precisely the function that versioning can fulfill. This paper presents versioning as a method to make research in the humanities and cultural sciences more trustworthy by providing researchers with greater knowledge about the origin and genesis of their data.

To date, the various approaches that have emerged over the last 15 years have not produced a universally accepted versioning standard for RDF data. The suitability and usefulness of these approaches for the digital humanities will be evaluated in the final section of the present paper (5), where typical research scenarios will demonstrate the different requirements for versioning mechanisms. First, however, I will address the extent to which versioning can contribute to the trustworthiness of data (2). This will be followed by a discussion of the various approaches to the versioning of RDF data (3), and of the question as to which archiving models are suitable for which application (4).

2 Versioning as a Dimension of Provenance

In 2000, Tim Berners-Lee launched his seminal idea of the Semantic Web Stack (Berners-Lee, 2000) (Figure1). Significantly, it is not only concerned with the development of technical standards to make information machine-understandable, but also incorporates social needs that arise out of dealing with the Semantic Web, with the layers of 'proof' and 'trust' representing essential preconditions for broad acceptance and use of shared content.

Since the debut of Berners-Lee's stack, many of the technologies featured in Figure1 have become W3C standards (XML, RDF(S), OWL, etc.). When it comes to 'proof' and 'trust,' however, the situation is quite different. As becomes clear from Berners-Lee's diagram, the task of the proof layer is to explain the conclusions drawn from the logical layer and to make their origin traceable – Sergei Sizov has referred to the former as "the layer of provenance knowledge" (Sizov, 2007, p. 94). It is this knowledge about resources and what has become of them that engenders trust in the Semantic Web.

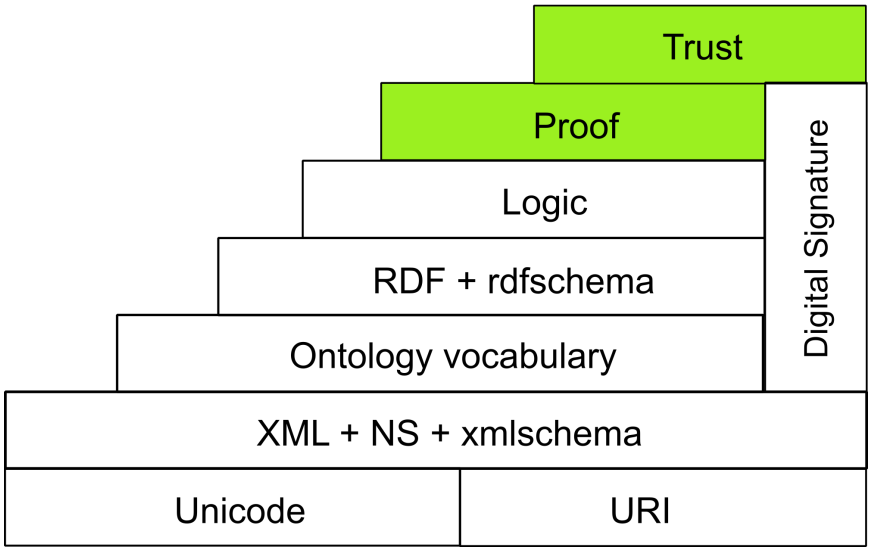

Figure 1: Semantic Web Stack with proof and trust layers highlighted

The World Wide Web Consortium (W3C) is an institution that promotes the development and implementation of internet-related standards. Between September 2009 and December 2010, the W3C sponsored the formation of a Provenance Incubator Group, whose goal was to define the phenomenon of provenance in the context of data management as comprehensively as possible. Their final report presents 14 dimensions of provenance – one of which is versioning, defined as "records of changes to an artifact over time and what entities and processes were associated with those changes" (Gil et al., 2010). Having emphasized that changes must be considered in context, the report goes on to explain that this kind of record is considered a dimension of provenance precisely because

> [d]ealing with evolution and versioning is a critical requirement for a provenance representation. As an artifact evolves over time, its provenance should be augmented in specific ways that reflect the changes made over prior versions and what entities and processes were associated with those changes (Gil et al., 2010).

Recording versions of digital resources to document change over time is an important part of describing provenance. Versioning enables us to refer to intermediate stages in the research process, but it also allows us to trace a 'version history' and to identify and analyze very specific changes between versions. In this respect, versioning functions as a mechanism of the proof layer and helps to foster confidence in digital resources.

3 RDF Versioning Approaches

RDF versioning can be implemented in different ways. Three different approaches are presented in this section, the first of which is reification (3.1), a way of attaching provenance information or change descriptions to individual triples. The concept of named graphs is then discussed as a possible alternative (3.2), before the final subsection addresses the extent to which traditional version control systems are suitable for the versioning of RDF graphs (3.3).

3.1 Reification

In RDF, it is possible to further describe each RDF statement through a built-in vocabulary.[1] This process is called reification, and describes the relationship between an instance of a triple and those resources to which the triple refers: "reification is a method of formally modeling a statement in such a way that it can actually be attached as a property to the new statement" (Powers, 2003, p. 69). A reified statement can also contain information on provenance (who made the statement and when), which strengthens confidence in the statement:

> [T]he key component of reification is the ability to make a statement and have the statement be treated as fact, without any implication that the contents of the statement are themselves facts. This has particular interest when it comes to trust (Powers, 2003, p. 78).

In order to version a dataset via reification, it is necessary to furnish individual triples or a set of triples with information on authors, a version number, or a timestamp. Ideally, the nature of the change itself is also described. For this purpose, separate vocabularies such as the Changeset Vocabulary by Tunnicliffe and Davis (2005-2009) have been developed,[2] which makes it possible to express an exact delta between two versions of a resource by means of two sets of triples (additions and removals). An example description in RDF/XML appears as follows:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/">
    <rdf:Description rdf:about="http://example.com/res#thing"/>
        <dc:title>Original Title</dc:title>
        <dc:description>A short description of this resource</dc:description>
```

[1] There is a rdfs:class called rdf:Statement, with the properties rdf:predicate, rdf:subject and rdf:object, https://www.w3.org/TR/rdf-schema/#ch_reificationvocab.

[2] http://purl.org/vocab/changeset

```
    </rdf:Description>
</rdf:RDF>
```

This example shows a resource with a description and a title. Then the title is modified:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/">
    <rdf:Description rdf:about="http://example.com/res#thing"/>
        <dc:title>New Title</dc:title>
        <dc:description>A short description of this resource</dc:description>
    </rdf:Description>
</rdf:RDF>
```

Describing this kind of modification as a reified statement would look like this:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:cs="http://purl.org/vocab/changeset/schema#">
    <cs:ChangeSet rdf:about="http://example.com/changesets#change">
        <cs:subjectOfChange rdf:resource="http://example.com/res#thing"/>
        <cs:createdDate>2006-01-01T00:00:00Z</cs:createdDate>
        <cs:creatorName>Anne Onymous</cs:creatorName>
        <cs:changeReason>Change of title</cs:changeReason>
        <cs:removal>
            <rdf:Statement>
                <rdf:subject rdf:resource="http://example.com/res#thing"/>
                <rdf:predicate rdf:resource="http://purl.org/dc/elements/1.1/title"/>
                <rdf:object>Original Title</rdf:object>
            </rdf:Statement>
        </cs:removal>
        <cs:addition>
            <rdf:Statement>
                <rdf:subject rdf:resource="http://example.com/res#thing"/>
                <rdf:predicate rdf:resource="http://purl.org/dc/elements/1.1/title"/>
                <rdf:object>New Title</rdf:object>
            </rdf:Statement>
        </cs:addition>
    </cs:ChangeSet>
</rdf:RDF>
```

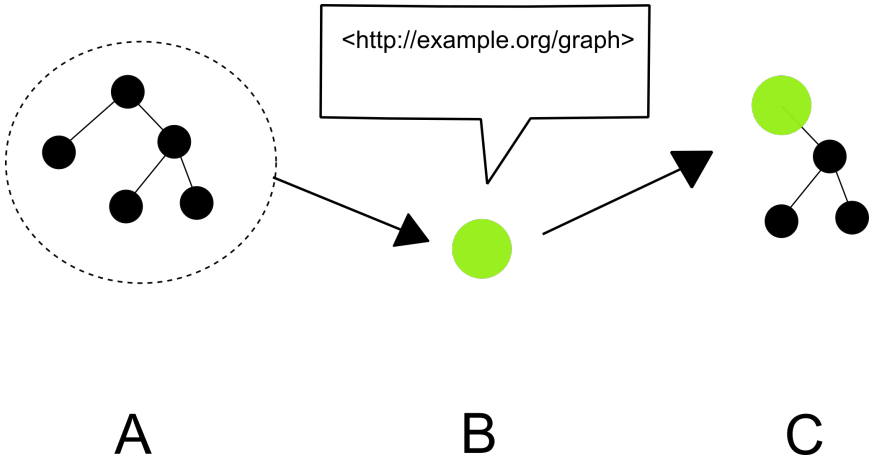

Figure 2: Named graph

While the changes are described precisely by the vocabulary, the change description ends up being much longer than the statement itself. Even if no exact description of changes is given, each reification of triples involves the addition of a whole statement with subject, predicate, and object (Seaborne and Davis, 2010). This syntactical overhead is a major reason why versioning is often implemented in a different way that reduces memory requirements (see section 4).

3.2 Named Graphs

The semantic concept of named graphs goes back to an essay by Carroll et al. (2005). One of their main concerns was to establish a framework that makes resources more trustworthy. When version 1.1 of the RDF standard was introduced in 2014, named graphs were included. Since then, it has been possible to name graphs and extend triple statements with an additional component (IRI):

> An RDF dataset is a collection of RDF graphs. All but one of these graphs have an associated IRI or blank node. They are called named graphs, and the IRI or blank node is called the graph name. The remaining graph does not have an associated IRI, and is called the default graph of the RDF dataset (Schreiber and Raimond, 2014).

Figure 2 shows a graph (A), which is assigned a name (B). This named graph then forms part of a new graph (C), which contextualizes the original graph and associates it with other statements. Compared to the mechanism of reification, a named RDF graph is easier to read and much more space efficient, which is precisely why current approaches to RDF versioning tend to use this type of information linkage. If a triple is modified, i.e. added or deleted, it

can be provided with an IRI, which in turn can be described by other statements.

3.3 Version Control Systems

Traditional version control systems have their uses, especially in software development. They store and manage text files which, when modified, lead to new versions with a unique identifier, a timestamp, and the author's name. It is always possible to determine who changed what and when: every file in the system has a version history, which allows versions to be compared. This comparison is done using diff algorithms, which detect any change to the file, be it in the addition of spaces or the correction of a spelling mistake. Each saved change automatically establishes a new version, allowing the restoration of each individual saved text state. But how useful is it in practical terms to manage graphs in line form?

Serialization techniques allow RDF graphs to be managed as text in line form with traditional version control systems. The best way to accomplish this is to use N-Triples notation, in which the individual triples of an RDF graph are written in one line. Because the N-Triples format employs neither prefixes nor truncated notation, the serialized result of the same RDF graph always looks the same. Sorted N-Triples provide a canonical representation of RDF that is easy to parse and serialize. However, this format has the disadvantage of being verbose and tedious to read.

The following example describes an entry from the Getty Art & Architecture Thesaurus[3] in N-Triples:

```
<http://vocab.getty.edu/aat/300343387>
<http://www.w3.org/2000/01/rdf-schema#label>
"national libraries (institutions)"@en .
<http://vocab.getty.edu/aat/300343387>
<http://purl.org/dc/terms/license>
<http://opendatacommons.org/licenses/by/1.0/> .
<http://vocab.getty.edu/aat/300343387>
<http://purl.org/dc/terms/created>
"2010-06-10T15:11:49"^^<http://www.w3.org/2001/XMLSchema#dateTime> .
```

By comparison, the same statements in Turtle appear as follows:

```
@prefix aat: <http://vocab.getty.edu/aat/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
```

[3] http://www.getty.edu/research/tools/vocabularies/aat/

```
@prefix dct: <http://purl.org/dc/terms/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema .

aat:300343387
rdfs:label "national libraries (institutions)"@en ;
dct:license <http://opendatacommons.org/licenses/by/1.0/> ;
dct:created "2010-06-10T15:11:49"^^xsd:dateTime .
```

Especially for small datasets, version management with traditional systems can be a viable choice (Meinhardt et al., 2015), but using them to version RDF graphs comes with several disadvantages:

1. If changes to the graphs are to be tracked, the differences between two graphs must be calculated. The result (delta) is large even for small changes (for example, even a minute change to the URI of the subject entails changes in all other lines that make statements about the subject).
2. As a consequence, the quality criteria which should be fulfilled by a diff algorithm are not met:

   > [T]he diff should construct a minimum set of changes to transform one version into the next one. Minimality is important because it captures to some extent the semantics that a human would give when presented with the two versions. It is important also in that more compact deltas provide savings in storage (Cobena et al., 2002).

3. If a change is made to the text that does not lead to a change in the graph, because, for example, a space is inserted before the period at the end of the statement, the dependence of delta on the serialization of the text means that a delta is calculated that indicates that a textual change has occurred.
4. This also means that it is impossible to materialize a past version of a given graph together with the delta – what is required is the graph in the exact same serialization that was used to create the delta.

4 RDF Archiving

The main challenges in versioning RDF data are the storage of versions, the performance of the archive, and the associated possibilities of information retrieval (Fernandez Garcia et al., 2018; Fernández et al., 2015). There are three major strategies for archiving versions: 'independent copies' (4.1), where a snapshot or a complete copy is stored; the 'change-based' approach (4.2), where only the changes to the graphs are recorded; and the 'time-based'

approach (4.3), which takes the lifetime of triples into account by archiving the validity period for statements. In real-world technical implementations, these three approaches to archiving frequently appear in combination, which helps to bring the requirements of the respective archive situation and the desired retrieval functionalities into line with available resources in terms of performance and storage capacity.

4.1 Independent Copies

An important advantage of the 'independent copies' strategy is the low technical effort required to implement the storage of data record versions. Each version is stored and managed as a new, isolated dataset. The DBpedia project,[4] initiated by the Freie Universität Berlin and Leipzig University in cooperation with OpenLink Software with the goal of extracting data from Wikipedia and making it available as Linked Open Data (LOD), has used this method to archive 18 versions of their entire data pool over the course of the regular updates undertaken between its initial release in 2007 and October 2016.

Wikidata,[5] a project of Wikimedia Germany, also archives independent datasets. The project started in 2012 with the objective of providing data that can be used by any Wikimedia project, including Wikipedia. Since 2015, data dumps have been made available on the Internet Archive, amounting to a total of 162 Wikidata versions by the time of writing this paper.[6] In order to make these versions queryable and comparable, each version could be stored as a graph in a triple store and provided with additional metadata concerning provenance.[7] This way of dealing with multiple versions is very well suited for playing back entire versions and querying individual versions.[8] However, the disadvantage of this approach is that it is very memory-intensive: the process of archiving each version is accompanied by an increasing number of duplicated triples, because there is a static core of unchanged triples that occurs in every version.

[4] http://wiki.dbpedia.org/

[5] https://www.wikidata.org/w/index.php?title=Wikidata:Main_Page&oldid=1086709037

[6] https://archive.org/details/wikimediadownloads

[7] The user Tbt (https://www.wikidata.org/wiki/Wikidata:History_Query_Service) is working on a tool to query past Wikidata versions. In the future, all deletions in a named graph and all additions in a separate named graph will be available for query. On the discussion page, the author writes the following: "This tool ingests data from the XML revision dumps, so it follows what is done for dumps regarding oversight. I do not know exactly what is done for XML dumps." Tpt (talk) 16:57, 9 April 2019 (UTC)

[8] On query requirements, (Fernandez Garcia et al., 2018; Fernández et al., 2015).

4.2 Change-Based Approach

The redundancies discussed in the previous subsection do not occur with change-based archiving. The amount of memory required is kept to a minimum. Only the changes (deltas) are archived, while static elements do not have to be repeated. The delta of each triple consists of the change description and the change relationship (delete or add).[9] Because triples, unlike text, can be localized without reference to lines, a version consists of a set of deleted triples and a set of added triples (Graube et al., 2014).[10]

Many practical features of version control systems can also be found in the implementation of systems adapted to RDF such as R&Wbase, a system proposed by Vander Sande et al. (2013) that is based on the core concept of distributed version control, which enables triple read and write. Here, various functionalities known from version control systems such as committing, merging, branching, and tagging are made available for collaborative work on RDF datasets. Commits, for example, are described using the PROV ontology, with each commit including a timestamp, the previous version, the name of the version just created, a title, and a responsible user. In effect, R&Wbase thus works as a separate versioning layer in a quad store.

In many respects, R43ples, the implementation developed by Graube et al. (2014), is very similar. It does, however, employ an additional triple store which acts as what we could call a 'versioning proxy.' As the application discussed before, the versions can be queried and updated via SPARQL – Graube et al. (2014) have introduced specific SPARQL keywords for this purpose, namely REVISION, BRANCH, and TAG.

Archives that store triple versions via a change-based approach are designed to record the changes to versions. When querying a specific state of the dataset at a certain point in time, the system's computational effort increases: first, all deltas up to the first entry (or the next full version/snapshot) must be recalculated, followed by the deltas from the reconstructed first version to the desired version. In order to cut down on the amount of processing power required, Im et al. (2012) introduced the concept of aggregated delta, which is independent from its predecessor version because it combines all change information in compressed form and as such reduces response time significantly.

[9] Barabucci 2016 defines delta as follows: "A delta [...] is a tuple of changes (C) and change relations (R) that describes how to transform the source document (S) into the target document (T)" (Barabucci et al., 2016, 50).

[10] The deltas are calculated from syntactic changes to the data. Changes in semantics (e.g. when a class is renamed) are much more complex to calculate – these are referred to as high-level deltas (Fiorelli et al., 2017, pp. 147-148). Some version control systems (e.g. Subversion, `https://subversion.apache.org/`) solely save changes.

4.3 Time-Based Approach

As mentioned above, there are also approaches that add a time component to the previously discussed storage forms (snapshot or delta) with the goal of optimizing system performance for very specific queries (e.g. the query for valid triples at a certain time or interval).

A time-based strategy can be implemented in two ways. One possibility is to assign to each triple, for as long as it exists, meta information in the form of a time stamp as each new version is created. However, with this approach, there is always write work for the system, since it also assigns a new time stamp to triples that remain unchanged. An alternative would be to only annotate triples when they are added or deleted, meaning that a maximum of just two data fields is added to the triple.

This implementation can be found, for example, in X-RDF-3X, a platform developed by Neumann and Weikum (2010) which adds two additional fields to the triple: one timestamp for creation and one for deletion, with the latter having a zero value for valid triple versions. The interval between the created and deleted timestamp represents the lifetime of the triple version. The state of a database at a certain point in time can thus be reconstructed by returning all triples for which the point in time falls within the corresponding lifetime intervals (Neumann and Weikum, 2010, p. 258) – a system which allows a quick representation of what has changed from one version to the next, because a change means that the respective triple must be recorded with a time stamp. This procedure is therefore also change-based, but the delta consists of timestamps as opposed to deleted and added triples in different named graphs.

There are also systems that combine all three of these archiving strategies in order to get the best of all worlds. In the implementation by Meinhardt et al. (2015), each version in the archive exists as a changeset that contains information about modifications at a certain point in time. The changeset contains at least one snapshot. If triples are added, they are stored as a delta to a snapshot with a time stamp; if a triple is deleted, a new snapshot is created, also carrying a time stamp. When a version is to be materialized, then the snapshot closest to the requested time is retrieved and the corresponding delta is added. The approach of Taelman et al. (2019) also combines snapshot and delta archiving. In addition, they rely on special indexing methods for enhanced efficiency in "evaluating queries at a certain version, between any two versions, and for versions" (Taelman et al., 2019, p. 4).

5 Use Cases in the Digital Humanities

The models for version storage discussed in the present paper allow for a wide range of query requirements to be met. But what kind of demands regarding the versioning of RDF or Linked Open Data exist within the context of the digital humanities?

Fiorelli et al. (2017) define the requirements for an RDF versioning system by distinguishing between user and developer. The user is primarily interested in being able to reference saved versions, whereas the developer wishes to track changes. In actual scholarly practice, however, the requirement profiles cannot be separated so neatly – in many cases, there is a need for both perspectives. This section presents three likely user scenarios to showcase the importance of versioning in the digital humanities, and to demonstrate how well the RDF versioning approaches discussed so far perform when confronted with real-life challenges. The first scenario addresses the problem of data consistency (5.1), scenario two deals with the issue of collaborative research (5.2), and scenario three is concerned with the specific nature of scholarly discourse in the humanities (5.3).

5.1 Version Reference and Data Consistency

Scenario 1: Occasional Data Updates for a Digital Scholarly Edition (DSE)

> I am the editor of a digital scholarly edition of account books. I modeled my data in RDF and published it. I did a statistical analysis of the data and displayed it to the users. Now I want to add another account book, but the evaluation of my analysis is based on a closed database. How can I make the current state of research accessible and still publish new RDF data?

On the one hand, this scenario describes a basic retrieval task, namely the retrieval of a specific version of the recorded data. On the other, there is also the need to query this past version, for example in order to keep statistical values verifiable. The 'independent copy' approach is perfectly adequate to fulfill these requirements, as each historical version dumps in an ordinary RDF memory. It is easy to provide full versions in the form of snapshots and to enrich these snapshots with additional metadata – for example, the PROV ontology can be used to create a version catalog or to provide additional provenance data, whereas different graphs (versions) can be queried via SPARQL.

A time-based strategy involving the annotation of individual triples with their validity range could also work well. Materializing a specific past version is the most computationally demanding task when using an approach that

only stores deltas, since all deltas must be re-calculated when requesting a full version. Assuming that the deltas are stored in such a way that they are only connected to the previous delta, the change chain is calculated back to the first complete version and then, in order to be able to materialize a certain later state, the deltas are calculated forward again up to the desired version. In this scenario, combined storage approaches that record both snapshots and deltas can help to keep the computational effort manageable.

5.2 Change Inspection and Evolution Tracking

Scenario 2: Collaborative Ontology Development

> I work for the National Library, and we want to develop a cross-institutional common ontology that is maintained collaboratively. With these collaborative processes, would it not be useful to track the changes and to analyze how and why they occurred?

This scenario focuses on the problem of collaborative editing. There is a need to query the changes that occurred between two or more versions. Here, functionalities provided by version control systems are expected – these involve merging branches, highlighting conflicts, quickly undoing changes, and so on. Yet a second requirement is also formulated, namely the tracking of very specific changes.

Calculating the difference between the versions of the datasets with an 'independent copy' approach requires significant computational assets given that the calculation of specific deltas takes place at query time. With a time-based approach, however, this task can be accomplished with little effort, provided that the time of adding and deleting the triple is available as meta information for each triple. In either case, the change-based implementations by Vander Sande et al. (2013) and Graube et al. (2014) discussed in subsection 4.2 are useful because they include practical functionalities of version control systems that can be queried using an extended SPARQL vocabulary. They also offer the possibility to use the 'commit' function to describe the changes in the collaborative process in more detail.

5.3 Qualitative/Quantitative Analysis of Data Evolution

Scenario 3: Analysis of Historical Topic Evolution in Wikidata

> I am a historian, and I want to analyze historical topic development in Wikidata from the beginning of Wikidata until today. I am interested in the evolution of knowledge and its representation. For instance, how are historical events and persons perceived and described? What are the primary topics, and how have certain aspects evolved and changed over time? How can I retrieve this kind of data?

In this scenario, the goal is to discover patterns that arise and develop over time. This results in specific requirements: not only should it be possible to compare past versions with each other, but the system should also allow users to query the changes that have taken place. To query information that is present in several versions, or specific changes that take place between them, is a task that is very difficult to accomplish with a straightforward 'independent copy' approach – it is far easier to query the validity of triples using a time-based solution. However, when it comes to tracking concept changes, the ability to query deltas is required. The approach developed by Meinhardt et al. (2015) makes it possible to search for changes with certain timestamps by using the MEMENTO protocol. The solutions proposed by Taelman et al. (2019) are another promising way to address this retrieval challenge.

6 Conclusion

When versioning data and archiving it for research purposes, its usability for future scholarship is a major concern. The choice of a specific versioning strategy depends on how often and how much our research data changes and, above all, on which information is to be made retrievable. This paper has illustrated the different requirements for versioning systems with user scenarios that are likely to occur in the context of the digital humanities. As has become clear, there are technically simple solutions to the problem of rendering RDF data and the changes that it is subjected to referenceable and retrievable. System requirements are bound to increase with a higher frequency of changes and mounting demands concerning their traceability, especially in collaborative research. For the collaborative development of graph data, versioning models are required that adopt functionalities from collaborative software development. Additional features permitting detailed analysis of changes over time are also desirable, and will empower scholars to undertake increasingly complex research projects as more and more RDF data is being published – and changed – in years to come.

As scholars working in the digital humanities, we are particularly interested in the origin and development of our research data. Versioning can be used to integrate and provide this critical information, whose importance for scholarly practice cannot be overstated. Moreover, versioning mechanisms ensure that our data can itself be used as an object of study. Not only does versioning create trust – it also enables the scholarly discourse of the future.

References

Barabucci, G., Ciancarini, P., Di Iorio, A., and Vitali, F. (2016). Measuring the Quality of Diff Algorithms: A Formalization. *Computer Standards*

& Interfaces, 46:52–65, DOI: 10.1016/j.csi.2015.12.005.

Berners-Lee, T. (2000). Semantic Web on XML. https://www.w3.org/2000/Talks/1206-xml2k-tbl/slide1-0.html.

Borgman, C. L. (2010). *Scholarship in the Digital Age. Information, Infrastructure, and the Internet*. MIT Press, Cambridge, MA, DOI: 10.7551/mitpress/7434.001.0001.

Carroll, J., Bizer, C., Hayes, P., and Stickler, P. (2005). Named Graphs, Provenance and Trust. In *WWW'05: Proceedings of the 14th international conference on World Wide Web*, pages 613–622. DOI: 10.1145/1060745.1060835.

Cobena, G., Abiteboul, S., and Marian, A. (2002). Detecting Changes in XML Documents. In *Proceedings 18th International Conference on Data Engineering*, pages 41–52. DOI: 10.1109/ICDE.2002.994696.

Fernández, J. D., Umbrich, J., Polleres, A., and Knuth, M. (2015). Towards Efficient Archiving of Dynamic Linked Open Data. In Debattista, J., d'Aquin, M., and Lange, C., editors, *Proceedings of the First DIACHRON Workshop on Managing the Evolution and Preservation of the Data Web*, volume 1377 of *CEUR Workshop Proceedings*, pages 34–49. http://ceur-ws.org/Vol-1377/.

Fernandez Garcia, J. D., Umbrich, J., Polleres, A., and Knuth, M. (2018). Evaluating Query and Storage Strategies for RDF Archives. *Semantic Web Journal*, http://epub.wu.ac.at/6488/.

Fiorelli, M., Pazienza, M. T., Stellato, A., and Andrea, T. (2017). Change Management and Validation for Collaborative Editing of RDF Datasets. *International Journal of Metadata, Semantics and Ontologies*, 12(2/3):142–154, DOI: 10.1504/IJMSO.2017.090783.

Gil, Y., Cheney, J., Groth, P., Groth, P., et al. (2010). Provenance XG Final Report. Technical report, W3C Incubator Group.

Graube, M., Hensel, S., and Urbas, L. (2014). R43ples: Revisions for triples - An Approach for Version Control in the Semantic Web. In *Proceedings of the 1st Workshop on Linked Data Quality co-located with 10th International Conference on Semantic Systems, SEMANTiCS*.

Im, D.-H., Lee, S.-W., and Kim, H.-J. (2012). A Version Management Framework for RDF Triple Stores. *International Journal of*

Software Engineering and Knowledge Engineering, 22(1):85–106, DOI: 10.1142/S0218194012500040.

Meinhardt, P., Knuth, M., and Sack, H. (2015). TailR: A Platform for Preserving History on the Web of Data. In *Proceedings of the 11th International Conference on Semantic Systems, ACM*, SEMANTICS '15, pages 57–64, New York, NY. Association for Computing Machinery, DOI: 10.1145/2814864.2814875.

Neumann, T. and Weikum, G. (2010). X-RDF-3X: Fast Querying, High Update Rates, and Consistency for RDF Databases. *Proc. VLDB Endowment*, 3(1/2):256–263.

Powers, S. (2003). *Practical RDF*. O'Reilly, Beijing/Cambridge.

Schreiber, G. and Raimond, Y. (2014). RDF 1.1 Primer. Technical report, W3C.

Seaborne, A. and Davis, I. (2010). Supporting Change Propagation in RDF. In *Proceedings of the W3C Workshop – RDF Next Steps*.

Sizov, S. (2007). What Makes You Think That? The Semantic Web's Proof Layer. *IEEE Intelligent Systems*, 22(6):94–99, DOI: 10.1109/MIS.2007.120.

Taelman, R., Vander Sande, M., and Van Herwegen, J. (2019). Triple Storage for Random-Access Versioned Querying of RDF Archives. *Web Semantics: Science, Services and Agents on the World Wide Web*, 54:4–28, DOI: https://doi.org/10.1016/j.websem.2018.08.001.

Tunnicliffe, S. and Davis, I. (2005-2009). Changeset Vocabulary. https://vocab.org/changeset/.

Vander Sande, M., Colpaert, P., Verborgh, R., Coppens, et al. (2013). R & Wbase: Git for Triples. In Bizer, C., Heath, T., Berners-Lee, T., Hausenblas, M., and Auer, S., editors, *Proceedings of the 6th Workshop on Linked Data on the Web*. http://ceur-ws.org/Vol-996/papers/ldow2013-paper-01.pdf.

Zimmerman, A. S. (2008). New Knowledge from Old Data: The Role of Standards in the Sharing and Reuse of Ecological Data. *Science, Technology, & Human Values*, 33(5):631–652, DOI: 10.1177/0162243907306704.