Collaborative Open Data versioning: a pragmatic approach using linked data.

Lorenzo Canova, Simone Basso, Raimondo Iemma, Federico Morando

Nexa Center for Internet & Society at Politecnico di Torino, DAUIN – Corso Duca degli Abruzzi 24, 10129 Torino (Italy), lorenzo.canova@polito.it, raimondo.iemma@polito.it, simone.basso@polito.it, federico.morando@polito.it

Abstract: *Most Open Government Data initiatives are centralised and unidirectional. For non trivial applications reusers make copies of the government datasets, so that they can curate their local copy of the data (e.g., by fixing errors). This situation is not optimal, it leads to duplication of efforts reducing the possibility of sharing. In this paper we describe a data publication pipeline that exports legacy-databases data into RDF and we investigate possible implementations of a feedback-channel. We show that reusers may want to merge changes at different stages of the pipeline; i.e. fixing errors to improve RDF triples as well as contributing better code for generating such triples, thus improving (open) data quality and therefore reusability and transparency. We discuss the features of the feedback-channel arguing that a full-fledged RDF versioning system is beneficial. We conclude reviewing existing RDF-versioning solutions and suggesting the next steps needed to implement a distributed RDF-versioning system.*

Keywords: *Public Sector Information, RDF versioning, Linked Open Data, Version control system.*

Acknowledgement: *This paper was drafted in the context of the Odino project. The authors acknowledge the support of the EU ERDF[1] fund ("POR FESR 07/13 - Measure L1.3 – Feasibility studies", application code: 270-166C) and the Piedmont Region. The authors are grateful to the project members for their support.*

1. Introduction

Most Open Government Data (OGD) initiatives are centralised and unidirectional; to simplify the job of reusers wishing to create visualisations and mockups, several open data portals implement Application Programming Interfaces (APIs) allowing reusers to download selected pieces of data. However, for non-trivial applications, many reusers often make copies of the governmental dataset so that they can curate their local copy of the data. In many cases, to enable reuse it is necessary to improve its quality (e.g., fixing errors and updating

[1] ERDF funds: http://ec.europa.eu/regional_policy/thefunds/regional/index_en.cfm

formats, which also enables cross-checking data for fostering government transparency), which, again, happens locally. Arguably, this situation is not optimal, since it leads to duplication of efforts and it reduces the possibility of sharing. The situation could be improved by asking governments to publish their dataset in standard formats, e.g., using the Resource Description Framework (RDF)[2], and by implementing feedback channels allowing improvements to be merged back into the original datasets.

Open Government Data can be made available in different ways and formats. A widely used reference to evaluate their reusability is the Five Star Open Data scale[3] by Tim-Berners Lee. In particular, "Linked Data" (the fifth star in the aforementioned scale) refers to a set of best practices for modelling and interconnecting information in a semantic[4] way. Linked data uses RDF as framework. RDF handles information as a network of semantic statements, called triples, consisting of subject, predicate, and object. Each information entity is referenced by an Internationalised Resource Identifier (IRI). Triples may be complemented by a fourth element, thus becoming a quadruple, that refers to a named graph. A named graph is a collection of triples grouped together and is identified by an IRI itself[5].

As of August 2014 the crawlable linked open datasets are more than a thousand[6], moreover from 2011 to 2014 the Linked Open Government datasets have grown by 306%, passing from 49 (17% of the total crawlable LOD) published Linked dataset to 199 (M. Schmachtenberg et al, 2014). They are published by hundreds of sources and can be accessed in several ways, such as, e.g.: issuing queries via SPARQL, de-referencing HTTP IRIs, or downloading data dumps and deploying them locally[7].

RDF facitilates the integration of several datasets. Moreover, since several different parties have write access[8], it is relatively easy to add new information in a decentralised fashion.

However, it is not always simple and intuitive to publish using RDF, since typically data is generated in other formats. For instance, to convert tabular data from legacy databases of public administrations into RDF, several steps need to be undertaken. This is one of the main reason why RDF, even if available from the year 2001, has not bee widely used since a few years ago, now it seems that it started being more and more adopted with a fast growing pace (M. Schmachtenberg et al, 2014).

The remainder of this paper is structured as follows. In section 2 we describe a possible data publication pipeline that translates legacy data into RDF, and we investigate possible implementations of feedback channels. In section 3 we show how RDF data could be versioned

[2] In this paper the reader might have a better comprehension is she is familiar with the concept at the base of RDF like: "triple", "IRIs", "SPARQL", "dereferencing HTTP URI", "named graph"and formats in which RDF can be saved. A good insight to this and more terminology is given by A. Ngonga Ngomo et al. (2014).

[3] FSOD: http://5stardata.info/

[4] „Semantic" in a data model means describing the meaning of the information within the data model. Semantics enable the capability to express information that permits to parties to interpret meaning from the instances, without the need to know the meta-model. For a broader definition see: http://en.wikipedia.org/wiki/Semantic_data_model

[5] A quadruple can be represented in this way: <graphname> <subject1> <predicate1> <object1>, where graphname is an IRI (or URI) that indicates to which set of triples (i.e. graph) is this triple (<subject1> <predicate1> <object1>) belonging to.

[6] Statistics on LOD: http://linkeddatacatalog.dws.informatik.uni-mannheim.de/state/#toc14.

[7] For example loading an RDF dump (that is a text file of triples) on a local triple store (like Virtuoso).

[8] i.e., through the use of IRIs, annotations of the same object can be made in multiple distributed datasets

and we review existing RDF version control system solutions. In section 4 we suggest further steps in order to implement and evaluate a collaborative distributed (Git-like[9]) versioning system.

2. Versioning and feedbacks in the Linked Open Data pipeline

In this section we discuss a possible pipeline for creating an RDF dataset from a legacy database, see section 2.1. Then, in section 2.2, we analyse where a versioning system could be implemented within this procedure. Finally, in section 2.3, we discuss the advantages of a full-fledged versioning system compared to a simple feedback procedure, and which aspects could enable a version control system for Linked Open Data (LOD).

2.1 The LOD pipeline

Linked Open Data have an enormous potential[10]. However, when data derives from legacy databases, the publication of LOD is not always immediate; frequently data comes from different sources and it needs to be gathered in a single file before proceeding with the conversion/translation into RDF triples (the so-called "triplification").

Different pipelines for LOD publication can be found[11], however in Figure 1, we take as example the common case in which data has been collected in some sort of legacy database and the user cannot control the procedure of data extraction from it. In this case, we can identify three main steps to publish RDF:

1. Extraction: data needs to be extracted from the database and put in a processable format in order to proceed with further steps. In this case (in Figure 1) we took CSV as target format for the extraction procedure, but it could also be done in other formats (e.g.: XML).
2. Transformation: data is converted from CSV to RDF file; different substeps can be implemented for this procedure.
    a. Cleaning the data[12]: data in CSV is easier to handle with standard data cleansing tools (e.g., Open Refine[13]) or scripts. Therefore it is recommended to clean the data before converting.
    b. Transform-mapping into JSON: even if it's not a necessary step, it is an incremental step in order to have an easier job for afterwards[14]. Moreover we

[9] Git is a revision control system. "Git-like" is intended as a decentralised version control system with emphasis on data integrity and non-linear workflows. For more information check this resource: http://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

[10] Some of the potentialities are explained the article by A. Ngonga Ngomo et al. (2014, p. 1-5)..

[11] In some cases data are collected and directly published in RDF. See: „*Publishing Linked Sensor Data*" (P.Barnaghi, et al., 2010), retrievable on: http://epubs.surrey.ac.uk/470673/1/sense2web.pdf

[12] Data cleaning in informatics is the process of correcting corrupt or inaccurate records. See: http://en.wikipedia.org/wiki/Data_cleansing

[13] Open Refine is a tool for working with messy data: cleaning it and also transforming it from one format into another. Source: http://openrefine.org/

[14] A good amount of documentation is available for converting JSON into triples. Here an article that explains the closeness between JSON and RDF: http://milicicvuk.com/blog/2014/08/26/can-json-and-rdf-be-friends/#q1

found useful the possibility to use a tool like gson[15] to automatically and easily convert special characters in UTF-8.

c. Assigning explicit semantics to the data: for this step a beforehand ontological study[16] should be done. Once the classes and properties to associate to the values of the original CSV have been decided, the conversion/translation into RDF triples is possible using, for example, tools like the Jena[17] library.

d. RDF: at the end of this process it is possible to create the RDF file in the format we like most (e.g.: RDF/XML, Turtle, RDF/JSON).

3. Staging and publishing:

a. Create the graph: once we have the RDF file we simply load it on our SPARQL endpoint.

b. Interlinking process: i.e., linking data to other resources that may describe the information being published. In practical terms, it may mean declaring a "SameAs"[18] with a DBpedia[19] resource, or other semantic resources (like SPC data[20] in Italy).

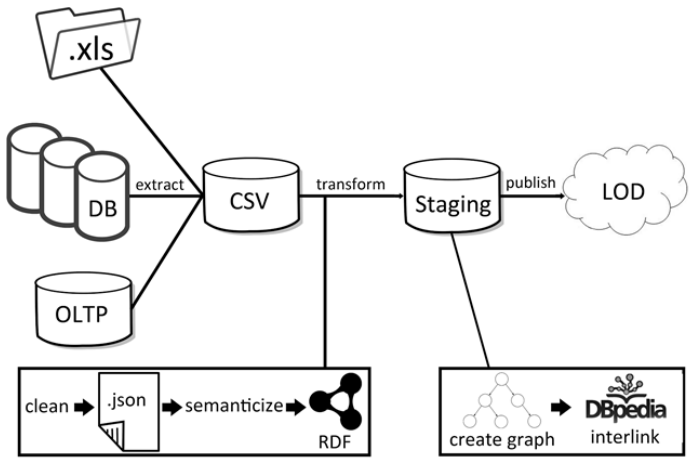

Figure 1: LOD publishing pipeline from legacy database (DB)

2.2 Version control system at different stages

In this paper, we intend version control as

> *"The management of changes to documents, computer programs, large web sites, and other collections of information" (Wikipedia contributors, 2014).*

[15] For detailed information on gson, please see: https://code.google.com/p/google-gson/

[16] Ontological study is intended as a beforehand study where classes and properties are defined in order describe the dataset domain (and therefore dataset values). Actually just a vocabulary could be defined. A vocabulary can be described as a light-weight ontology, i.e. a collection of URIs with a described meaning. Source: http://semanticweb.org/wiki/Ontology

[17] Jena library is part of the Jena software that is „A free and open source Java framework for building Semantic Web and Linked Data applications". Source: https://jena.apache.org/

[18] Definition of SameAs: https://schema.org/sameAs

[19] DBpedia: http://dbpedia.org/About

[20] SPC data: http://spcdata.digitpa.gov.it/index.html

The major function of a version control system is to record changes that could be rolled back, or that could be made by different developers (authors in the case of data) and could be merged into a single base (e.g., the so-called master branch).

Depending on their objective, reusers may want to merge changes at different stages of the pipeline shown in section 2.1; e.g., one could fix errors contributing improved RDF triples as well as contributing better code for generating such triples. We identify three distinct phases in which versioning could have potential benefits:

1. During the transformation of data: versioning source code for cleaning and triplifying data (step 2 in pipeline explained in section 2.1) would result particularly helpful in specific frameworks where a generalist cleansing tool (e.g.: a tool that removes common errors like double spacing) would not be enough. It would allow user and publishers to collaborate in creating domain specific tools for certain datasets.
2. In data feedback loops: in this case a version control system would be used to track the provenance and changes made by users on single triples allowing feedback loops.
3. For forking[21] and merging data: it may happen that some data needs to be enriched with complementary data. If the publisher could not directly do it, users could create specific improved forks of the dataset. The publisher could later merge them with the official version he publishes (in order to create higher-value certified datasets), or simply publicize specific high-value forks made by user communities. For merging data some domain specific issues may arise, the Open Contracting Data Standard specifies an approach to merging data within the schema specification and documentation[22].

The potential for reuse would therefore be maximum not only when the code used for generating RDF is versioned, but also when the RDF triples themselves are versioned.

2.3 Simple feedback channel VS full-fledged versioning system: benefits and possible applications

Nowadays, in most of the open data initiatives it has not been possible for users to contribute to the released data. Usually it is not possible neither to send direct feedback (like flag for errors or comments), nor to update, extend, or correct the existing data (M.Vander Sande, et al., 2013).

In some cases, however, a simple feedback procedure is set up using alternative channels (e.g., trivially, e-mail, comments, etc.). For simple tasks it is already a useful procedure. For example, let's say that some data is published in a good way, but there is an ambiguity in the meaning of a certain metadata, if a "contacts" form is provided, it is possible to have a clarification and probably the ambiguity will be resolved also for other users, via FAQ, or correcting the metadata[23]. This

[21] Forking data means to take a copy of the (semantic) database and start independent development/improvement on it, creating a distinct and separate piece of the database. In Open Data it should be permitted without prior permission without violating any copyright law. Adapted version of: http://en.wikipedia.org/wiki/Fork_(software_development).

[22] For further information see:
http://ocds.open-contracting.org/standard/r/1__0__RC/en/implementation/merging/

[23] This situation happened for real in analysing OpenCoesione data on: http://www.opencoesione.gov.it/

kind of approach, even if it is easy to implement, is rather inefficient because of poor data management.[24]

In order to overcome these issues, Linked data could be the enabler for "Open Data Ecosystems"[25], in which open data feedback loops are a fundamental part of the open data lifecycle (Pollock, 2011). Feedback loops would allow users to improve (i.e. patch) data, in order to obtain, for instance, better quality and more valuable data. Moreover, if a full-fledged version control system on the published data is implemented, forks and merges become possible.

A Git-like (see footnote 9) distributed version control system allowing forking, experimenting and merging is supposed to improve the following aspects:

1. Data accuracy: public data is frequently released in a raw format and doesn't have a high accuracy. It is usually common to find errors, duplications and missing data. The quality improvement steps could be made by users and then merged with the official validated version published by a central authority.
2. Completeness and Richness (complementary data): besides resolving accuracy issues, with a versioning system, users could also add complementary data to the one provided by the government, thereby implementing a fully working collaborative loop.
3. Timeliness - coordinating teams: in some cases, the frequency of update of open government data is insufficient to preserve data meaningfulness, and/or not advertised. With a versioning system it could be possible to have a unofficial but always up-to-date set of data (when the source of the data permits it). Moreover Public Sector bodies could validate and merge with the official version up-to-date branches[26] made by other users, cutting de facto the update procedure cost.
4. Comparability of today's data versus yesterday's data: being able to rollback modification would allow historical analysis.
5. Trustworthiness of data (provenance): a fully working versioning system would provide provenance of the data branches and revisions in a way that the consumer could choose which source to trust and which version of the data to use.
6. Opportunities for communities that improve data: versioning with provenance would permit reachability and more visits to communities that furnish, by means of forks[27], well improved data.
7. Natural selection of datasets: high value datasets would be incrementally improved. There would be a better quality where it is really needed.

[24] There are some virtuous cases in which open government data versioning (to tabular data) has already been implemented. Even though in these cases there is not a formalized feedback procedure, forks and merges are already possible. See for example the city of Chicago's initiative: https:// github.com/Chicago/. See also: Vander Sande, et al., 2013, p.2.

[25] A full description of the Open Data Ecosystem that is based on feedback loops is given by Rufus Pollock (2011) on http://blog.okfn.org/2011/03/31/building-the-open-data-ecosystem/

[26] Branches are created through "branching ". Branching is the duplication of an object under revision control in a way that modifications can happen in parallel along different branches. In this paper branching and forking have the same meaning.

[27] Forks are obtained by means of forking, see note 21.

By improving the explained aspects, we can arguably say that a full-fledged RDF versioning system is preferable to a simple feedback channel, even though a simple feedback channel is more practical for many simple applications.

An interesting aspect is that of incentives, from both of the sides (reusers and government/publisher), i.e., what will motivate reusers to improve the accuracy of a public dataset and contribute that back to government, and what will motivate government to carry out the quality checks and reviews necessary to merge in the changes suggested.

As regards re-users incentives, a similar problem has been analysed by G.Kuk et al. (2011), where is stated that the best improvements and services based on published open dataset and then shared with the community have been developed in hack day events, starting a virtuous circle for the improved dataset (services were made upon that specific dataset and more people got involved in keeping it improved and up-to-date). However only for few interesting datasets these improvements were made (and kept updated during time), while the majority remained ignored. We believe that some sort of gamification, or something like assessing the Five Star Open Data Engagement[28] (from the government side), could help improving also the not considered datasets, however, as stated in 7 on the aforementioned list, this "natural selection" seems unavoidable.

From the government's incentive point of view we believe that it is a "just win" situation, government can make the choice to merge community data with its own, in order to provide better and updated data with less effort than doing all the work with its own resources, however they could also keep publishing their data without caring about the community versions.

3. RDF versioning methods

Version control systems are available for development of software source code, for relational databases, for websites and also for CSV, JSON and XML data (*Dat project*[29] offers a working alfa version for versioning CSV/XML/JSON files), however for RDF data it isn't a deeply explored area.

In this paragraph we show what the main general approaches for versioning RDF data are (section 3.1), and the solutions found to solve this issue (section 3.2).

3.1 General aspects

There are two main ways to implement revisioning of databases (or domain models generally) (Pollock, 2010):

1. Copy on Write (CoW): it is used, in its simplest way, in order to have a full copy of the database at each version. Usually it is made, more efficiently, by restricting the copy-on-write only to the changed objects.
2. Diffs: it stores diffs between versions and possibly a full version of the model at a given point in time (snapshot).

Usually in both cases a set of metadata is bound to the revision (or changeset) object, those are:

- timestamp and/or unique identifier of the change;

[28] Five Star Open Data Engagement: http://www.opendataimpacts.net/engagement/

[29] Official site of *Dat*: http://dat-data.com/ , github repository https://github.com/maxogden/dat

- description of the change (log message);
- author of the change;
- digital signature[30].

For expressing changesets in triples the PROV-O ontology[31], which is basically today's standard for expressing provenance, might be useful.

For implementing versioning in RDF data, CoW and Diff need different approaches:

- RDF versioning with CoW: in this case we would need a way to reference entities (triples in our case) and a way for putting objects in "deleted" state. For referencing triples we could use the "Context value"[32] (in which we could put the commit), or use reification[33].
- RDF versioning with Diff: a given version of the graph would be obtained composing diffs. This implementation of versioning is more efficient storage-wise, but it's harder to use (and implement) compared to CoW.

3.2 Existing solutions

Table 1 shows a brief qualitative literature review of existing RDF versioning solutions. Some of them have a proper name indicated in the field "Name", the ones that did not have an assigned name are indicated by an asterisk and are associated to a recognisable identifier[34].

In the field "Peculiarities" we annotated the characteristics that distinguish a certain solution from another. The next 5 fields/criteria are described as follows:

1. Low Storage overhead: in this field we evaluated (qualitatively) how heavy, storage-wise, was creating a new version of a triple;
2. Easy access to versions: this criteria basically valuates two aspects 1) if it is easy to access different versions and 2) if versions can be queried separately;
3. Available implementation: for this field we checked if any working implementation of the descripted versioning method was available;
4. Permits branching: this criteria is taking in consideration if it is possible to enable branching and merging (thus also versioning in a distributed fashion);
5. Compliance with standards: in this case as "standards" we meant semantic standards. Specifically the questions we asked our-selves to evaluate this characteristic were 1) how could an external machine know with what version is dealing with? And 2) how could the latest version be accessed by a general (not instructed) user/machine? The value in this field are "Low" (as for low compliance), "Med" (stands for medium compliance) and "High".

In last filed, "Specific Issues", we spotted the issues that may arise using a certain system for versioning semantic data that were not included in the 5 general fields in the aforementioned list.

[30] The digital signature is usually implemented through a „Sign-off", that is a line at the end of the commit message. It certifies who the author of the commit is and its main purpose is tracking of who did what, especially with patches.

[31] PROV-O ontology: http://www.w3.org/TR/prov-o/

[32] The context value is the fourth element of a triple (or in this case a quadruple). For further information on the context value please see the description of N-Quads on: http://sw.deri.org/2008/07/n-quads/

[33] For further information: https://jena.apache.org/documentation/notes/reification.html

[34] The identifier is constructed with 1) a characteristic of the solution and 2) Name of main author.

Table 1: State of art in RDF data version control systems.

Name	Peculiarities	Low storage overhead	Easy access to versions	Available implementation	Permits branching	Compliance with standards	Specific Issues
SemVersion (M. Völkel and T. Groza, 2006.)	CVS[35]-based RDFS and OWL versioning system. Provides support for blank nodes	No	No	No	Yes	Low	Branches are not supported at query time. Formalisation of deltas between two versions is unknown.
Partial graphs - Schandl* (B. Schandl, 2010)	Version control in the context of replicating partial RDF graphs. Optimised for devices with limited computing power and memory. SVN-like[36].	No	No	Yes	Yes	Med	Provenance is not available.
Patches Version Control - Cassidy* (S. Cassidy and J. Ballantine, 2007)	Darcs' theory of patches, a version is a sequence of patches. Each patch is identified by a named graph.	No	No	No	Yes	Med	Formalisation of patches is unknown.
Custom DB versioning - Im* (D.-H. Im et al., 2012)	No snapshots. Only use of original version and consecutive versions' delta to reduce storage space. Supports parallelisation. Version is constructed with SQL queries. Custom relational DB to store triples.	Yes	Yes	Yes	Yes	Low	System dependent on the database. Interoperability issues with existing triple stores.
Temporal RDF	Tracking information over time ("time	Yes	No	No	No	Low	Changes are not bundled in a

[35] CVS stands for Concurrent Versioning System. It is a client-server (free) software revision control system. See: http://en.wikipedia.org/wiki/Concurrent_Versions_System

[36] SVN is the abbreviation of "Subversion". It is a software versioning and revision control system. For further information refer to: https://subversion.apache.org/

(C. Gutierrez et al., 2007)	labelling"). In this work a syntax is defined for incorporating temporality into standard RDF graphs.						semantic way. No query for temporal RDF well compatible with SPARQL.
Atomic changes and reification – Auer (S. Auer and H. Herre, 2007)	Atomic changes to RDF graphs annotated in reified statements of the original data.	Yes	Yes	Yes	Yes	Low	Specific operations not well integrated in the current Semantic Web environment.
Apache Marmotta - KiWi Versioning module (The Apache Software Foundation, 2014)	Implementation of a Linked Data Platform. Tracks changes to resources and the whole repository and identifies the source (provenance) of certain triples. It creates snapshots of the repository that are "known to be good".	Yes	Yes	Yes	No	Med	Reverting changes has not yet been implemented.
R&Wbase (Sande et al., 2013)	Use of git-like method (but no actual use of Git). Made with diffs, supports branches and parallelisation. Commits are stored in quadruples' context value. Separately accessible versions resolved at query time. Requires support for quads.	Yes	Yes **	Yes	Yes ***	Med	Need to maintain line order of each file. No support for deleted blank nodes. ***Only part of the system is modelled semantically (use of hash tables). Implementation not compatible with latest Virtuoso[37] version. **Slow to access different versions.
R43ples (Graube et al.,	Version control on a graph level. Use of Revision Management	Yes	Yes	Yes	Yes	Med/ High	Special keywords in SPARQL

[37] OpenLink Virtuoso is a SQL-ORDBMS and Web Application Server hybrid (aka Universal Sever) that provides SQL, XML, and RDF data management. Virtuoso provides a Triple Store accessible via SPARQL. Definition taken from: http://www.w3.org/2001/sw/wiki/OpenLink_Virtuoso

2014)	Ontology (PROV-O plus specific terms). Implementation: it is a SPARQL proxy, is a Java application. Performance tests have been made (Good only to medium-size dataset).						queries. No support for blank-nodes. Extensive use of named graph.

* the project that don't have a proper name are associated to a recognisable identifier composed by: 1) A characteristic of the solution and 2) The name of the main author.

3.3 Discussion

Natively for RDF a standard versioning system implementation has not been specified, however, as shown, different solution have been proposed. Between these, since we want machines to understand versioned content (in order to have automatized processes for merging, forking etc.), we must only take in consideration semantically-compliant[38] methods. If the data is versioned for example, like in the solution of Im et al. (2012), with specific relational databases, it would be hard to interpret the versions for an external user (or machine). Moreover for "SemVersion", "Patches Version Control – Cassidy" and "Temporal RDF" no implementation is provided.

Between the solutions with an available implementation and that permit to have a distributed versioning system, "Atomic changes and reification - Auer" is not well integrated in the current semantic web environment (due to the extensive use of reified statements), and Apache Marmotta doesn't give us any information on how its versioning system could be distributed (thus providing parallelisation and branching), while "Partial graphs – Schandl" doesn't consider to track provenance (fundamental for a distributed versioning system).

A working group of the W3C has also proposed some best practices for implementing versioning and implemented a domain-specific versioning method for versioning linked data[39](not tracked in Table 1), by providing the version information inside the URI, however we believe that this method is somewhat in contrast with the "cool URI" paradigm[40] and, implementing this solution, has a low compliance with (semantic) standards (as described in the aforementioned list in paragraph 3.2).

The only two solutions that could version triples in a distributed and semantically compliant way are R&Wbase and R43ples. However there are still some issues in the usage of those two: there are performance problems and there is no support for blank-nodes, moreover R&Wbase is not compatible with the latest Virtuoso version and is quite slow in accessing different versions.

M. Völkel et al. (2006) proposed some smart solution to handle blank nodes, however, practically (not having stable IRIs) they still remain hard to track in a RDF versioning system.

[38] By semantically-compliant version control system we mean that it implements versioning in a way that the commit is understandable by machines simply with a SPARQL query (and not e.g.: by parsing the URI).

[39] The W3C working group approach to versioning: http://www.w3.org/2011/gld/wiki/Best_Practices_Discussion_Summary#Versioning

[40] How cool URI should be built: http://www.w3.org/TR/cooluris/#semweb

4. Conclusions and future works

In this work we have shown how a versioning system for RDF could be beneficial. We also identified criteria to evaluate different versioning systems; a version control system for LOD should be compliant with the Semantic Web standards and should permit branching and merging in order to be machine interpretable and enhance a collaborative environment. Between the analysed solutions, only two meet these two macro requirements, thus allowing a feedback loop and also the forking of data. Right now, however, there is still the need to propose a concrete and standard solution for versioning RDF data with a fully working and semantically-compliant versioning system in order to enable all the benefits described in paragraph 2.3.

We have also shown that versioning would also be beneficial if it was collaborative in the upstream procedures (in the transformation phase), in order to facilitate cleaning procedures. We haven't found any tool or framework that would easily allow these two types of versioning together (upstream on the code and downstream on the triples). However, this effect could be obtained combining different approaches at different stages of the pipeline.

As future work, first there is the need to address the issues in the solutions proposed by Graube and Vander Sande (i.e.: handling blank nodes, performance problems, compliance with existing tools like Virtuoso). Afterwards it would be interesting to implement a framework that would permit upstream and downstream versioning in a Git-like manner.

References

Cassidy and J. Ballantine (2007). *Version control for RDF triple stores*. ICSOFT (ISDM/EHST/DC) '07, pages 5–12.

M. Vander Sande, A. Dimou, P. Colpaert, E. Mannens, R. Van de Walle (2013). *Linked Data as enabler for Open Data Ecosystems.* Open Data on the Web 23 - 24 April 2013, Campus London, Shoreditch.

R. Pollock (2011). *Building the open data ecosystem*. retrieved on December 1, 2014, from: http://blog.okfn.org/2011/03/31/building-the-open-data-ecosystem/

R. Pollock (2010). *Versioning / Revisioning for Data, Databases and Domain Models: Copy-on-Write and Diffs*. retrieved on December 1, 2014 at:

http://rufuspollock.org/2010/09/10/versioning-revisioning-for-data-databases-and-domain-models-copy-on-write-and-diffs/.

M. Völkel and T. Groza (2006). *Semversion: Rdf-based ontology versioning system*. In Proceedings of the IADIS International Conference WWW/Internet 2006 (ICWI), page 44.

B. Schandl (2010). *Replication and versioning of partial RDF graphs*. In Proceedings of the 7th international conference on The Semantic Web: research and Applications - Volume Part I, ESWC'10, pages 31–45, Berlin, Heidelberg. Springer-Verlag.

D.-H. Im, S.-W. Lee, and H.-J. Kim. A version management framework for RDF triple stores. international journal of software engineering and knowledge engineering, 22(01):85–106, 2012.

C. Gutierrez, C. Hurtado, and A. Vaisman (2007). *Introducing time into RDF*. IEEE Transactions on Knowledge and Data Engineering, 19(2):207–218.

S. Auer and H. Herre (2007). *A versioning and evolution framework for RDF knowledge bases*. In Perspectives of Systems Informatics, page 55–69. Springer.

M. Vander Sande, P. Colpaert, R. Verborgh, S. Coppens, E. Mannens, and R. Van de Walle (2013). *R&Wbase: git for triples*. In Proceedings of the 6th Workshop on Linked Data on the Web.

Apache Software Foundation (2014). Apache Marmotta: KiWi versioning. Retrieved on December 1, 2014 from: http://marmotta.apache.org/kiwi/versioning.html

Graube, Markus, Stephan Hensel, and Leon Urbas (2014). *R43ples: Revisions for Triples.* LDQ 2014, 1st Workshop on Linked Data Quality Sept. 2, 2014, Leipzig, Germany.

Wikipedia contributors (2014) . *Revision control*. Wikipedia, The Free Encyclopedia. December 15, 2014, 20:45 UTC. Retrieved on December 16, 2014. Available at: http://en.wikipedia.org/w/index.php?title=Revision_control&oldid=638259054.

A. Ngonga Ngomo , S.Auer, J. Lehmann, A.Zaveri (2014), *Introduction to Linked Data and Its Lifecycle on the Web*. Reasoning on the Web in the Big Data Era, DOI: 10.1007/978-3-319-10587-1_1 , page 1-99. Springer.

M. Schmachtenberg, C. Bizer, H. Paulheim (2014*). Adoption of the Linked Data Best Practices in Different Topical Domains*, The Semantic Web – ISWC 2014, pp 245-260, DOI 10.1007/978-3-319-11964-9_16. Springer International Publishing

G.Kuk, T.Davies (2011), *The Roles of Agency and Artifacts in Assembling Open Data Complementarities*, Thirty Second International Conference on Information Systems, Shanghai 2011

About the Authors

Lorenzo Canova

Lorenzo Canova holds a MSc in Industrial engineering from Politecnico di Torino and is a research fellow of the Nexa Center for Internet & Society since November 2014. His main research interests are in Open Governement Data, government transparency and linked data.

Simone Basso

Simone Basso is a research fellow of the Nexa Center for Internet & Society since 2010. At the Nexa Center he leads the Neubot and the MorFEO projects on network performance, network transparency and network neutrality. His main research interests are network performance, network neutrality, TCP, security, Internet traffic management, peer to peer networks, and streaming. He studied at Politecnico di Torino from which he received the B.Sc. degree (in 2006), the M.Sc. degree (in 2009) and the PhD degree (in 2014).

Raimondo Iemma

Raimondo Iemma holds a MSc in Industrial engineering from Politecnico di Torino. He is PhD candidate in Management, production and design at Politecnico di Torino. He performs research and policy support in the field of Open Government Data and innovation within (or fostered by) the Public Sector. He is Managing Director & Research Fellow at the Nexa Center for Internet & Society.

Federico Morando

Federico Morando is an economist, with interdisciplinary research interests at the intersection between law, economics and technology. He holds a Ph.D. in Institutions, Economics and Law from the Univ. of Turin and Ghent. He is the Director of Research and Policy of the Nexa Center for Internet & Society. From Dec. 2012, he leads the Creative Commons Italy project.