Dynamic Provenance for SPARQL Updates using Named Graphs

Harry Halpin
World Wide Web Consortium
hhalpin@w3.org

James Cheney
University of Edinburgh
jcheney@inf.ed.ac.uk

Abstract

The (Semantic) Web currently does not have an official or de facto standard way exhibit provenance information. While some provenance models and annotation techniques originally developed with databases or workflows in mind transfer readily to RDF, RDFS and SPARQL, these techniques do not readily adapt to describing changes in dynamic RDF datasets over time. Named graphs have been introduced to RDF motivated as a way of grouping triples in order to facilitate annotation, provenance and other descriptive metadata. Although their semantics is not yet officially part of RDF, there appears to be a consensus based on their syntax and semantics in SPARQL queries. Meanwhile, updates are being introduced as part of the next version of SPARQL. In this paper we explore how to adapt the dynamic copy-paste provenance model of Buneman et al. [2] to RDF datasets that change over time in response to SPARQL updates, how to represent the resulting provenance records themselves as RDF using named graphs, and how the provenance information can be provided as a SPARQL end-point.

1 Introduction

It is becoming routine to publish scientific and governmental data on the Web as RDF (the Resource Description Framework, a W3C standard for structured data on the Web). In doing so, it is crucial not to lose track of its provenance, including both its derivation process and changes to it over time. There are many different kinds of provenance that possibly could be associated with RDF. There is a difference between *static* provenance for processes that create new artifacts from existing ones, versus *dynamic* provenance that describes how artifacts have evolved over time. Second, there is a difference between provenance for *atomic* artifacts that expose no internal structure as part of the provenance record, versus provenance for *collections* or other structured artifacts. The workflow community has largely focused on static provenance for atomic artifacts, whereas much of the work on provenance in databases has focused on static or dynamic provenance for collections (e.g. tuples in relations).

The workflow community has largely focused on declaratively describing causality or derivation steps of processes to aid repeatability for scientific experiments, and these requirements have been a key motivation for the Open Provenance Model (OPM) [12, 11], a vocabulary and data model for describing processes including (but certainly not limited to) runs of scientific workflows. While important in its own right, and likely to form the basis for a W3C standard interchange format for provenance, OPM does not address the semantics of the provenance represented in it — one could in principle use it either to represent static provenance for processes that construct new data from existing artifacts, or to represent dynamic provenance that represents how data change over time. Most applications of OPM, however, seem to focus on static provenance, and so it may require further extension to handle dynamic provenance. Conversely, many elements of the OPM vocabulary may not be necessary for describing changes to RDF data over time.

It is an open question on how to best access provenance data for RDF. Currently provenance techniques are not easily usable to the vast majority of developers and other people working in open data. If we want RDF to become a leading format for open data, having an easy and straightforward manner to access provenance data could be crucial. We believe that dynamic provenance techniques should be as easy to use as version control. Version control is a classic and well-studied problem for information systems ranging from source code management systems to temporal databases and data archiving. Version control systems such as CVS, Subversion, and Git have a solid track record of tracking versions and (coarse-grained) provenance for text (e.g. source code) over time.

Recent work on provenance in relational databases has aimed precisely at "provenance as version control". Foundational work on provenance for queries distinguishes between where-provenance, which is the "locations in the source databases from which the data was extracted," and why-provenance, which is "the source data that had some influence on the existence of the data" [3]. There is less work considering provenance for updates; our previous work focused on simple atomic update operations: insertion, deletion, and copy [2]. A provenance-aware database should support queries that permit users to find the ultimate or proximate 'sources' of some data, explore its change history over time, or apply data provenance techniques to understand why a given part of the data was inserted or deleted by a given update. A simple approach that records the full provenance of every part of the data during every update would likely lead to an explosion of database size, so current research is studying ways of optimizing provenance storage and new formal models for relational data [2].

How can we track provenance for dynamic RDF data using current components of the Semantic Web? Static provenance techniques have now been investigated for RDF, including provenance-tracking for SPARQL queries (see [15, 10]) and for RDFS inferences (see [4, 8]) over RDF datasets. Some of this work considers updates, particularly Flouris et al. [8], who consider the problem of how to maintain provenance information for RDFS inferences when tuples are inserted or deleted, using the coherence semantics. In this paper we consider a provenance model for SPARQL queries and updates to data stores involving named graphs, whose purpose is to provide a record of how the raw data in a dataset has changed over time, and then tackle the problem of recording and providing (queryable) access to the detailed change history of an RDF graph that is updated over time via SPARQL updates.

Our hypothesis is that a simple vocabulary, composed of insert, delete, and copy operations as introduced by Buneman et al. [2], along with explicit identifiers for update steps, versioning relationships, and metadata about updates provides a flexible format for data provenance on the Semantic Web. The harder problem of devising a combined vocabulary for describing both static and dynamic provenance remains; nevertheless our proposal sheds some light on the issues. A primary advantage of our methodology is it keeps the changes to raw data separate from the changes in metadata, so legacy applications will continue to work and the cost of storing and providing access to provenance can be isolated from that of the raw data.

We will briefly review named graphs and their relationship to versioning and provenance, then introduce SPARQL queries and updates, and finally describe our proposals for versioning and provenance-tracking for RDF graphs and SPARQL updates.

2 Named graphs for dynamic provenance

From an early stage it was recognized that provenance management in RDF requires the ability to refer to triples or graphs. Hence, there is unofficial yet widely implemented support for *named graphs*, where each graph G is identified with a *name URI* [5]. The idea of using named graphs for provenance is not new, but previously has been restricted to making and propagating trust measures [9] or authorship assertions/signatures [5] through inferences or queries. We believe that the named graph approach is also well-suited to recording detailed provenance traces for dynamic data.

Our proposal is that the provenance information should be accessible from the name URI of a named graph by asking for the provenance content associated with the graph. Obviously, the provenance store for each graph should be accessible, ideally using Linked Data principles, from the name URI of the named graph. Moreover, named graphs can be used to represent both past versions of a graph and changes to the graph in order to support both versioning and dynamic provenance. In practice on the Web of Data, the graph name is usually the URI used to access the the RDF graph or the domain name of the server. For example, a group of RDF triples from DBPedia about Paris may be named by `http://www.dbpedia.org/data/Paris`. One could imagine that past versions of this graph might be named `http://www.dbpedia.org/data/Paris_v1`, etc., with additional URIs representing the updates that have been applied in a machine-readable format, along with conventional provenance metadata about the updates such as author, time, and the text of the update.

However, one problem with minting a new name URI for each past version of the graph would be that there would soon be an overwhelming number of name URIs, with one for each change to the graph. While they could still be connected to each other via the provenance data, a higher-level alternative would be to use query parameters to qualify the base URI. So one could have `http://www.dbpedia.org/data/Paris?version=1` to retrieve the first version, `http://www.dbpedia.org/data/Paris?version=2` to retrieve the second, and so on. A "date" parameter to retrieve by date would also be useful, such as `http://www.dbpedia.org/data/Paris?date=2010-08-30T21%3A33%2A50`.
To get the provenance metadata, one could then add a special "provenance" parameter like `http://www.dbpedia.org/data/Paris?provenance&version=2` and `http://www.dbpedia.org/data/Paris?provenance&date=2010-08-30T21%3A33%2A50`.

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?book ?who
WHERE { ?book dc:creator ?who }
```

Figure 1: SPARQL Query Example.

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
DELETE DATA {
 GRAPH <http://example.com/bookStore> {
  <http://example.com/book3>
    dc:creator "Harry Halpin"
 }
}
INSERT DATA {
 GRAPH <http://example.com/bookStore> {
  <http://example.com/book3>
    dc:creator "Henry Story"
 }
}
```

Figure 2: SPARQL Update Example

```
GET /sparql/?query=EncodedQuery \
            &provenance-date=2010-08-30 \
            &http://www.example/bookstore
Host: www.example
User-agent: sparql-client/0.1
```

Figure 3: Asking for provenance using HTTP GET

Note that multiple name URIs constructed with different parameters may result in the same graph.Furthermore, the same conventions could also be added to SPARQL Protocol for RDF [6] to allow easy access of RDF as shown in an example explained below in Section 3.

3 Strawman Example

Our approach is complementary to techniques for publishing or disseminating versioned Web resources (e.g. the Memento protocol of Van de Sompel et al. [16]) or accompanying static provenance or metadata (e.g. Coppens et al. [7]). Our advantage is to give a formal semantics that could be used to optimize this provenance, and a more simple query-parameter based method for accessing provenance that does not require new HTTP headers and subsequent changes to server and client software.

Suppose we discover that the author of a book is wrong in an RDF graph, and it needs to be changed. This is transmitted via this use of SPARQL Update to the end point of the named graph for the bookstore, `http://example.com/bookStore`, as given in Figure 1. Then another user-agent wishes to query (using the query in Figure 2) for the name of the book before the edits were made on August 30th 2010. The user transmits this via HTTP GET, specifying the necessary provenance date of August 30th 2010 in Figure 3. Then the SPARQL endpoint gives response with a return date using a provenance element in Figure 4.

```
HTTP/1.1 200 OK
Date: Fri, 01 July 2011 20:55:12 GMT
Server: Apache/1.3.29 (Unix) PHP/4.3.4 DAV/1.0.3
Connection: close
Content-Type: application/sparql-results+xml
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
<provenance date="2001-30-26T21:32:52" />
  <head>
    <variable name="book"/>
    <variable name="who"/>
  </head>
  <results distinct="false" ordered="false">
    <result>
    <binding name="book">
      <uri>http://www.example/book/book5</uri>
    </binding>
   <binding name="who">Harry Halpin</binding>
  </result>
...
  </head>
</provenance>
```

Figure 4: SPARQL Response for Provenance Information

4 SPARQL Background

Queries Let Lit be a set of literals (e.g. strings), let Id be a set of resource identifiers, and let Var be a set of variables usually written $?X$. We write $\mathsf{Atom} = \mathsf{Lit} \cup \mathsf{Id}$ for the set of atomic values, that is literals or ids. The syntax of core algebra for SPARQL discussed in [1] is as follows:

$$
\begin{array}{rcl}
A,B,C & ::= & l \in \mathsf{Lit} \mid \iota \in \mathsf{Id} \mid ?X \in \mathsf{Var} \\
t & ::= & \langle A\ B\ C \rangle \\
C & ::= & \{t_1,\ldots,t_n\} \mid \texttt{GRAPH}\ A\ \{t_1,\ldots,t_n\} \mid C\ C' \\
R & ::= & \texttt{BOUND}(?x) \mid A = B \mid R \wedge R' \mid R \vee R' \mid \neg R \\
P & ::= & C \mid P\ .\ P' \mid P\ \texttt{UNION}\ P' \mid P\ \texttt{OPT}\ P' \\
 & \mid & P\ \texttt{FILTER}\ R \\
Q & ::= & \texttt{SELECT}\ ?\vec{X}\ \texttt{WHERE}\ P \mid \texttt{CONSTRUCT}\ C\ \texttt{WHERE}\ P
\end{array}
$$

Here, C denotes basic graph (or dataset) patterns that may contain variables; R denotes conditions; P denotes patterns, and Q denotes queries. We do not distinguish between subject, predicate and object components of

triples, so this is a mild generalization of [1], since real SPARQL does not permit literals to appear in the predicate position or as the name of a graph in the GRAPH A $\{P\}$ pattern. We also do not consider blank nodes, which pose complications especially when updates are concerned. Another point to note is that we allow named graph patterns only as part of basic patterns. The semantics of core SPARQL algebra (from [1]) is given in the appendix.

Updates We will employ a simplified core language for atomic updates, based upon [14]:

$$\begin{array}{rcl} U & ::= & \texttt{INSERT}\ \{C\}\ \texttt{WHERE}\ P \mid \texttt{DELETE}\ \{C\}\ \texttt{WHERE}\ P \\ & \mid & \texttt{LOAD}\ g\ \texttt{INTO}\ g' \mid \texttt{CLEAR}\ g \mid \texttt{CREATE}\ g \mid \texttt{DROP}\ g \end{array}$$

The SPARQL Update working draft specifies that transactions consisting of multiple updates should be applied atomically, but leaves some semantic questions unresolved, such as whether aborted transactions have to rollback partial changes. It also does not specify whether updates in a transaction are applied sequentially (as in most imperative languages), or using a snapshot semantics (as in most database update languages). Both alternatives pose complications, so in this paper we focus on transactions consisting of single atomic updates, leaving the general case for future work.

5 Provenance model

A single SPARQL update can read from and write to several named graphs (and possibly also the default graph). For simplicity, we restrict attention to the problem of tracking the provenance of updates to a single (possibly named) RDF graph. All operations may still use the default graph or other named graphs in the dataset as sources. The general case can be handled using the same ideas as for a single anonymous graph, only with more bureaucracy to account for versioning of all of the named graphs managed in a given dataset.

We model the provenance of a single RDF graph G that is updated over time as a set of history records, including a special graph named `prov_G` and additional auxiliary named graphs such as `G_v0`,...,`G_vn` and `G_u1`...,`G_um`. Intuitively, `G_vi` is the named graph showing G's state in version i and `G_ui` is another named graph showing the triples inserted into or deleted from G by update i.

The provenance graph `prov_G` includes several kinds of nodes and edges:

- `G_vi upd:nextVersion G_vi+1` edges that show the sequence of versions;
- nodes `u1`,...,`un` representing the updates that have been applied to G, along with a `upd:type` edge linking to one of `upd:insert`, `upd:delete`, `upd:load`, `upd:clear`, `upd:create`, or `upd:drop`.
- For all updates except create, an `upd:input` edge linking `ui` to `G_vi`.
- For all updates except drop, an `upd:output` edge linking `ui` to `G_vi+1`.
- For insert and delete updates, an edge `ui upd:data G_ui` where `G_ui` is a named graph containing the triples that were inserted or deleted by `ui`.
- Edges `ui upd:source n` linking each update to each named graph `n` that was consulted by `ui`. For an insert or delete, this includes all graphs that were consulted while evaluating P (note that this may only be known at run time); for a load update, this is the name of the graph whose contents were loaded; create, drop and clear updates have no sources.
- Additional edges from `ui` providing metadata (such as author, commit time, log message, or the source text of the update) for the update; possibly using a standard vocabulary such as Dublin Core, or using OPM-style agent nodes and `wasControlledBy` edges. We omit the details of this metadata.

Note that this representation does not directly link triples in a given version to places from which they were "copied". However, it does provide enough information to recover this on request. Moreover, if we retain the source text of the update statements performed by each update, as well as all intermediate versions of the graph, we can trace backwards through the update sequence to identify places where triples were inserted or copied into or deleted from the graph.

This is a high-level, logical model of the provenance of a graph as it evolves over time, along with all of its intermediate versions. It is not a concrete proposal for how to store or query this graph information efficiently. We expect that sharing techniques similar to those used by version control systems or temporal databases can be used to store the sequence of versions efficiently; once this is done, the contents of the named graphs `G_ui` that store inserted or deleted triples can be represented efficiently by just storing the insert or delete statements themselves and recovering the graphs lazily on demand. However, we have not implemented this model and developing a practical implementation is future work.

6 Provenance semantics

For queries, we consider a simple form of provenance which calculates a set of named graphs "consulted" by the query. (For the purposes of this paper, this is an ad hoc, syntactic notion.) This could be viewed as a simple form of why-provenance, analogous to determining the names of the relations mentioned in a database query.

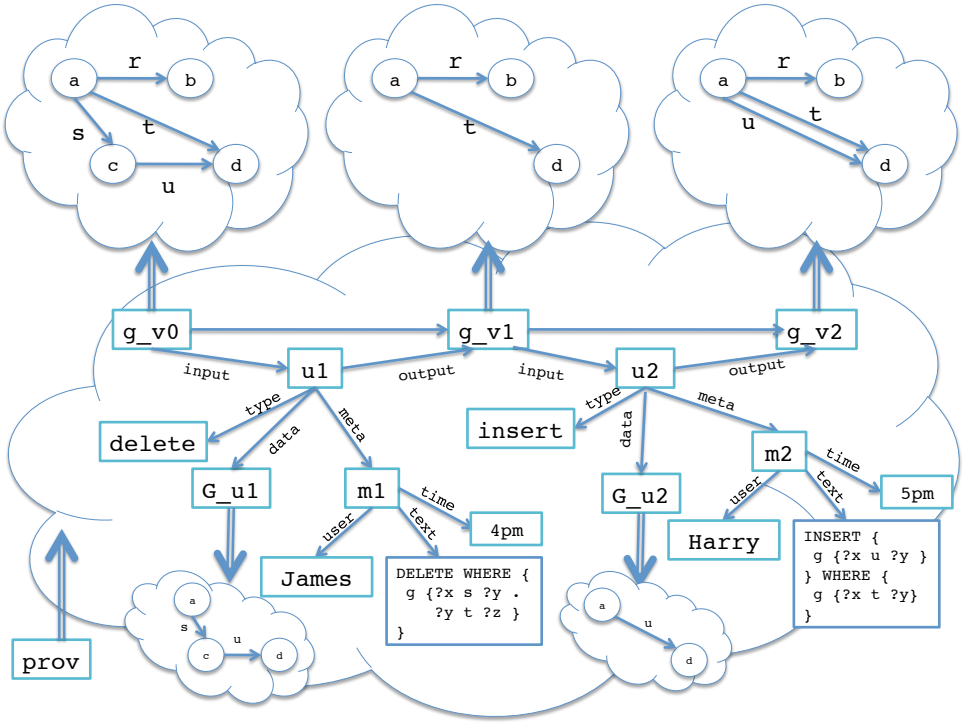

Figure 5: Example provenance record

Note, however, that unlike in a relational language, the names of the graphs consulted by a query are dependent on the data, since a pattern such as GRAPH $?X$ $\{\langle a\ b\ c\rangle\}$ can consult any graph that happens to contain $\langle a\ b\ c\rangle$.

We define the provenance of an atomic update by translation to a sequence of updates that, in addition to performing the requested updates to a given named graph, also constructs some auxiliary named graphs and triples in a special named graph for provenance information called *prov*.

We consider only special cases of insert and delete operations that target a single, statically known, named graph g; full SPARQL Updates (according to the current graph) can simultaneously update several named graphs along with the default graph.

A graph creation CREATE g is translated to

CREATE g;
CREATE g_v_0;
INSERT DATA {GRAPH *prov* {
  $\langle g$ version $g\_v_0\rangle, \langle g$ current $g\_v_0\rangle$,
  $\langle u_1$ type create$\rangle, \langle u_1$ output $g\_v_0\rangle$,
  $\langle u_1$ meta $m_i\rangle$, (metadata)
}}

A drop operation (deleting a graph) DROP g is handled as follows, symmetrically to creation:

DROP g;
DELETE WHERE {GRAPH *prov* {$\langle g$ currentg_v_i }}};
INSERT DATA {GRAPH *prov* {
  $\langle u_i$ type drop$\rangle, \langle u_i$ input $g\_v_i\rangle$,
  $\langle u_i$ meta $m_i\rangle$, (metadata)
}}

where g_v_i is the current version of g. Note that since this operation deletes g, after this step the URI g no longer names a graph in the store; it is possible to create a new graph named g, which will result in a new sequence of versions being created for it. The old chain of versions will still be linked to g via the version edges, but there will be a gap in the chain.

A clear graph operation CLEAR g is handled as follows:

CLEAR g;
DELETE WHERE {GRAPH *prov* {$\langle g$ currentg_v_i }}};
INSERT DATA {GRAPH *prov* {
  $\langle g$ version $g\_v_0\rangle, \langle g$ current $g\_v_0\rangle$,
  $\langle u_i$ type clear$\rangle, \langle u_i$ input $g\_v_i\rangle$,
  $\langle u_i$ output $g\_v_{i+1}\rangle, \langle u_i$ meta $m_i\rangle$,
  (metadata)
}}

A load graph operation LOAD h INTO g is handled as follows:

LOAD h INTO g;
DELETE WHERE {GRAPH *prov* {$\langle g$ currentg_v_i }$\rangle$};
INSERT DATA {GRAPH *prov* {
$\langle g$ version $g\_v_{i+1}\rangle, \langle g$ current $g\_v_{i+1}\rangle$,
$\langle u_i$ type load$\rangle, \langle u_i$ input $g\_v_i\rangle$,
$\langle u_i$ output $g\_v_{i+1}\rangle, \langle u_i$ source $h_j\rangle$,
$\langle u_i$ meta $m_i\rangle$, (metadata)
}}

where h_j is the current version of h.

An insertion INSERT {GRAPH g {C}} WHERE P is translated to a sequence of updates that creates a new version and links it to URIs representing the update, as well as links to the source graphs identified by the query provenance semantics and a named graph containing the inserted triples:

CREATE g_u_i;
INSERT {GRAPH g_u_i {C}} WHERE P;
INSERT {GRAPH g {C}} WHERE P;
CREATE g_v_{i+1};
LOAD g INTO g_v_{i+1};
DELETE DATA {GRAPH *prov* {$\langle g$ *current* $g\_v_i\rangle$}};
INSERT DATA {GRAPH *prov* {
$\langle g$ version $g\_v_{i+1}\rangle, \langle g$ current $g\_v_{i+1}\rangle$,
$\langle u_i$ input $g\_v_i\rangle, \langle u_i$ output $g\_v_{i+1}\rangle$,
$\langle u_i$ type insert$\rangle, \langle u_i$ data $g\_u_i\rangle$
$\langle u_i$ source $S_1\rangle, \ldots, \langle u_i$ source $S_m\rangle$,
$\langle u_i$ meta $m_i\rangle$, (metadata)}}

where $s_1, \ldots, s_m$ are the source graph names of P.

A deletion DELETE {GRAPH g {C}} WHERE P is handled similarly to an insert, except for the update type annotation.

7 Conclusion

Provenance is a challenging problem for RDF. While some progress has been made on provenance and annotation for RDFS inferences and SPARQL queries, so far there has not been work on provenance for SPARQL Updates, partly because the update language itself is work in progress. In this paper we have outlined an approach to the problem drawing on similar work in database archiving and copy-paste provenance, and sketched how this approach can be incorporated into existing protocols for accessing RDF datasets. We hope this will contribute to discussion of how to standardize descriptions of changes to RDF datasets, and possibly provide a way to translate changes to underlying (e.g. relational or XML) databases to RDF representations.

References

[1] ARENAS, M., GUTIERREZ, C., AND PEREZ, J. On the semantics of SPARQL. In *Semantic Web Information Management: A Model Based Perspective*, 1st ed. Springer, 2009.

[2] BUNEMAN, P., CHAPMAN, A., AND CHENEY, J. Provenance management in curated databases. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data* (New York, NY, USA, 2006), SIGMOD '06, ACM, pp. 539–550.

[3] BUNEMAN, P., KHANNA, S., AND TAN, W. Why and where: A characterization of data provenance. In *ICDT* (2001), no. 1973 in LNCS, pp. 316–330.

[4] BUNEMAN, P., AND KOSTYLEV, E. Annotation algebras for RDFS. In *SWPM* (2010).

[5] CARROLL, J. J., BIZER, C., HAYES, P., AND STICKLER, P. Named graphs. *Web Semant. 3* (December 2005), 247–267.

[6] CHARBONEAU, D., AND FEIGENBAUM, L. SPARQL 1.1 protocol for RDF. W3C Working Draft, January 2010. http://www.w3.org/TR/2010/WD-sparql11-protocol-20100126/.

[7] COPPENS, S., MANNENS, E., VAN DEURSEN, D., HOCHSTENBACH, P., JANSSENS, B., AND VAN DE WALLE, R. Publishing provenance information on the web using the Memento datetime content negotiation. In *LDOW* (2011).

[8] FLOURIS, G., FUNDULAKI, I., PEDIADITIS, P., THEOHARIS, Y., AND CHRISTOPHIDES, V. Coloring RDF triples to capture provenance. In *Proceedings of the 8th International Semantic Web Conference* (Berlin, Heidelberg, 2009), ISWC '09, Springer-Verlag, pp. 196–212.

[9] HARTIG, O. Querying trust in rdf data with tsparql. In *Proceedings of the 6th European Semantic Web Conference on The Semantic Web: Research and Applications* (Berlin, Heidelberg, 2009), ESWC 2009 Heraklion, Springer-Verlag, pp. 5–20.

[10] LOPES, N., POLLERES, A., STRACCIA, U., AND ZIMMERMANN, A. AnQL: SPARQLing up annotated RDFS. In *Proceedings of the 9th international semantic web conference - Part I* (Berlin, Heidelberg, 2010), ISWC'10, Springer-Verlag, pp. 518–533.

[11] MOREAU, L. Provenance-based reproducibility in the semantic web. *Journal of Web Semantics* (2011). To appear.

[12] MOREAU, L., CLIFFORD, B., FREIRE, J., FUTRELLE, J., GIL, Y., GROTH, P., KWASNIKOWSKA, N., MILES, S., MISSIER, P., MYERS, J., PLALE, B., SIMMHAN, Y., STEPHAN, E., AND VAN DEN BUSSCHE, J. The open provenance model core specification (v1.1). *Future Generation Computer Systems 27*, 6 (June 2011), 743–756.

[13] SCHENK, S., GEARON, P., AND PASSANT, A. SPARQL 1.1 Update. W3C Working Draft, October 2010. http://www.w3.org/TR/2010/WD-sparql11-update-20101014.

[14] UDREA, O., RECUPERO, D. R., AND SUBRAHMANIAN, V. S. Annotated RDF. *ACM Trans. Comput. Logic 11* (January 2010), A10.

[15] VAN DE SOMPEL, H., SANDERSON, R., NELSON, M. L., BALAKIREVA, L. L., SHANKAR, H., AND AINSWORTH, S. An HTTP-based versioning mechanism for linked data. In *LDOW* (2010).