Dynamic Provenance for SPARQL Updates using Named Graphs

Harry Halpin
World Wide Web Consortium
Massachusetts Institute of Technology
Cambridge, MA, USA
hhalpin@w3.org

James Cheney
School of Informatics
University of Edinburgh
Edinburgh, United Kingdom
jcheney@inf.ed.ac.uk

ABSTRACT

While the (Semantic) Web currently does have a way to exhibit static provenance information in the W3C PROV standards, the Web does not have a way to describe dynamic changes to data. 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. In this paper we explore how to adapt the dynamic copy-paste provenance model of Buneman et al. [1] 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 in a manner compatible with W3C PROV, and how the provenance information can be provided as a SPARQL query. The primary contribution is a semantic framework that enables the semantics of SPARQL Update to be used as the basis for a 'cut-and-paste' provenance model in a principled manner.

Keywords

SPARQL Update, provenance, versioning, RDF, semantics

1. OVERVIEW

Our hypothesis is that a simple vocabulary, composed of insert, delete, and copy operations as introduced by Buneman et al. [1], along with explicit identifiers for update steps, versioning relationships, and metadata about updates provides a flexible format for dynamic provenance on the Semantic Web. A primary advantage of our methodology is it keeps the changes to raw data separate from the changes in provenance 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.

2. PROVENANCE SEMANTICS

A single SPARQL update can read from and write to several named graphs (and possibly also the default graph). Any graph that records the insertion and deletion of triples from a given graph is considered a *provenance graph* for the given graph. The general concept is that in a fully automated process one should be able to re-construct the state of the given graph at any time from its provenance graph by following the history records for each update operation tracked by the provenance graph.

C denotes basic graph (or dataset) patterns that may contain variables; R denotes conditions; P denotes patterns, and Q denotes queries. A graph store $\mathcal{D} = (G, \{g_i \mapsto G_1 \ldots, g_n \mapsto G_n\})$ consists of a default graph G_0 together with a mapping from names g_i to graphs G_i.

We model the provenance of a single RDF graph that is updated over time as a set of history records, including the special provenance graph named *prov* which keeps track of auxiliary named graphs such as `G_v0`,. . . ,`G_vn` and `G_u1`. . . ,`G_um` that store the precise triples changed in each update (although they do not store the entire graph) along with associated metadata. 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.

For queries, we consider a simple form of provenance which calculates a set of named graphs "consulted" by the query. Unlike in a relational language, the names of the graphs consulted by a query are dependent on the data, since a pattern such as $\texttt{1}\ ?X\ \{\langle a\ b\ c\rangle\}$ can consult any graph that happens to contain $\langle a\ b\ c\rangle$. The set of sources of a pattern or query is computed as follows:

$$\begin{aligned}
\mathcal{S}[\![C]\!]^{\mathcal{D}}_{G} &= \bigcup\{\mathrm{names}(\mu(C)) \mid \mu \in [\![C]\!]^{\mathcal{D}}_{G}\} \\
\mathcal{S}[\![P_1\ \texttt{1}\ P_2]\!]^{\mathcal{D}}_{G} &= \mathcal{S}[\![P_1]\!]^{\mathcal{D}}_{G} \cup \mathcal{S}[\![P_2]\!]^{\mathcal{D}}_{G} \\
\mathcal{S}[\![P_1\ \texttt{1}\ P_2]\!]^{\mathcal{D}}_{G} &= \mathcal{S}[\![P_1]\!]^{\mathcal{D}}_{G} \cup \mathcal{S}[\![P_2]\!]^{\mathcal{D}}_{G} \\
\mathcal{S}[\![P_1\ \texttt{1}\ P_2]\!]^{\mathcal{D}}_{G} &= \mathcal{S}[\![P_1]\!]^{\mathcal{D}}_{G} \cup \mathcal{S}[\![P_2]\!]^{\mathcal{D}}_{G} \\
\mathcal{S}[\![P\ \texttt{1}\ R]\!]^{\mathcal{D}}_{G} &= \mathcal{S}[\![P]\!]^{\mathcal{D}}_{G} \\
\mathcal{S}[\![\texttt{1}\ ?\vec{X}\ \texttt{1}\ P]\!]^{\mathcal{D}} &= \mathcal{S}[\![P]\!]^{\mathcal{D}} \\
\mathcal{S}[\![\texttt{1}\ C\ \texttt{1}\ P]\!]^{\mathcal{D}} &= \mathcal{S}[\![P]\!]^{\mathcal{D}}
\end{aligned}$$

where the auxiliary function names(C) collects all of the graph names occurring in a ground basic graph pattern C:

$$\begin{aligned}\text{names}(\{t_1,\ldots,t_n\}) &= \emptyset\\ \text{names}(\mathtt{1}\ A\ \{t_1,\ldots,t_n\}) &= \{A\}\\ \text{names}(C\ C') &= \text{names}(C)\cup\text{names}(C')\end{aligned}$$

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 (the history records) and triples in a special named graph for provenance information called *prov* (the provenance graph). We detail how provenance information should be attached to each SPARQL Update operation u.

- A graph creation $\mathtt{1}\ g$ is translated to

  $\mathtt{1}\ g$;
  $\mathtt{1}\ g\_v_0$;
  $\mathtt{1}\ \mathtt{1}\ \{\mathtt{1}\ \mathit{prov}\ \{$
  $\langle g\ \mathtt{version}\ g\_v_0\rangle, \langle g\ \mathtt{current}\ g\_v_0\rangle,$
  $\langle u_1\ \mathtt{type\ create}\rangle, \langle u_1\ \mathtt{output}\ g\_v_0\rangle,$
  $\langle u_1\ \mathtt{meta}\ m_i\rangle$, (metadata)
  $\}\}$

- A drop operation (deleting a graph) $\mathtt{1}\ g$ is handled as follows, symmetrically to creation:

  $\mathtt{1}\ g$;
  $\mathtt{1}\ \mathtt{1}\ \{\mathtt{1}\ \mathit{prov}\ \{\langle g\ \mathtt{current}\ g\_v_i\rangle\}\}$;
  $\mathtt{1}\ \mathtt{1}\ \{\mathtt{1}\ \mathit{prov}\ \{$
  $\langle u_i\ \mathtt{type\ drop}\rangle, \langle u_i\ \mathtt{input}\ g\_v_i\rangle,$
  $\langle u_i\ \mathtt{meta}\ m_i\rangle$, (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.

- A clear graph operation $\mathtt{1}\ g$ is handled as follows:

  $\mathtt{1}\ g$;
  $\mathtt{1}\ \mathtt{1}\ \{\mathtt{1}\ \mathit{prov}\ \{\langle g\ \mathtt{current}\ g\_v_i\rangle\}\}$;
  $\mathtt{1}\ \mathtt{1}\ \{\mathtt{1}\ \mathit{prov}\ \{$
  $\langle g\ \mathtt{version}\ g\_v_{i+1}\rangle, \langle g\ \mathtt{current}\ g\_v_{i+1}\rangle,$
  $\langle u_i\ \mathtt{type\ clear}\rangle, \langle u_i\ \mathtt{input}\ g\_v_i\rangle,$
  $\langle u_i\ \mathtt{output}\ g\_v_{i+1}\rangle, \langle u_i\ \mathtt{meta}\ m_i\rangle,$
  (metadata)
  $\}\}$

- A load graph operation $\mathtt{1}\ h\ \mathtt{1}\ g$ is handled as follows:

  $\mathtt{1}\ h\ \mathtt{1}\ g$;
  $\mathtt{1}\ \mathtt{1}\ \{\mathtt{1}\ \mathit{prov}\ \{\langle g\ \mathtt{current}\ g\_v_i\rangle\}\}$;
  $\mathtt{1}\ \mathtt{1}\ \{\mathtt{1}\ \mathit{prov}\ \{$
  $\langle g\ \mathtt{version}\ g\_v_{i+1}\rangle, \langle g\ \mathtt{current}\ g\_v_{i+1}\rangle,$
  $\langle u_i\ \mathtt{type\ load}\rangle, \langle u_i\ \mathtt{input}\ g\_v_i\rangle,$
  $\langle u_i\ \mathtt{output}\ g\_v_{i+1}\rangle, \langle u_i\ \mathtt{source}\ h_j\rangle,$
  $\langle u_i\ \mathtt{meta}\ m_i\rangle$, (metadata)
  $\}\}$

  where h_j is the current version of h.

- An insertion $\mathtt{1}\ \{\mathtt{1}\ g\ \{C\}\}\ \mathtt{1}\ P$ 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:

  $\mathtt{1}\ g\_u_i$;
  $\mathtt{1}\ \{\mathtt{1}\ g\_u_i\ \{C\}\}\ \mathtt{1}\ P$;
  $\mathtt{1}\ \{\mathtt{1}\ g\ \{C\}\}\ \mathtt{1}\ P$;
  $\mathtt{1}\ g\_v_{i+1}$;
  $\mathtt{1}\ g\ \mathtt{1}\ g\_v_{i+1}$;
  $\mathtt{1}\ \mathtt{1}\ \{\mathtt{1}\ \mathit{prov}\ \{\langle g\ \mathtt{current}\ g\_v_i\rangle\}\}$;
  $\mathtt{1}\ \mathtt{1}\ \{\mathtt{1}\ \mathit{prov}\ \{$
  $\langle g\ \mathtt{version}\ g\_v_{i+1}\rangle, \langle g\ \mathtt{current}\ g\_v_{i+1}\rangle,$
  $\langle u_i\ \mathtt{input}\ g\_v_i\rangle, \langle u_i\ \mathtt{output}\ g\_v_{i+1}\rangle,$
  $\langle u_i\ \mathtt{type\ insert}\rangle, \langle u_i\ \mathtt{data}\ g\_u_i\rangle$
  $\langle u_i\ \mathtt{source}\ s_1\rangle, \ldots, \langle u_i\ \mathtt{source}\ s_m\rangle,$
  $\langle u_i\ \mathtt{meta}\ m_i\rangle$, (metadata)$\}\}$

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

- A deletion $\mathtt{1}\ \{\mathtt{1}\ g\ \{C\}\}\ \mathtt{1}\ P$ is handled similarly to an insert, except for the update type annotation.

  $\mathtt{1}\ g\_u_i$;
  $\mathtt{1}\ \{\mathtt{1}\ g\_u_i\ \{C\}\}\ \mathtt{1}\ P$;
  $\mathtt{1}\ \{\mathtt{1}\ g\ \{C\}\}\ \mathtt{1}\ P$;
  $\mathtt{1}\ g\_v_{i+1}$;
  $\mathtt{1}\ g\ \mathtt{1}\ g\_v_{i+1}$;
  $\mathtt{1}\ \mathtt{1}\ \{\mathtt{1}\ \mathit{prov}\ \{\langle g\ \mathit{current}\ g\_v_i\rangle\}\}$;
  $\mathtt{1}\ \mathtt{1}\ \{\mathtt{1}\ \mathit{prov}\ \{$
  $\langle g\ \mathtt{version}\ g\_v_{i+1}\rangle, \langle g\ \mathtt{current}\ g\_v_{i+1}\rangle,$
  $\langle u_i\ \mathtt{input}\ g\_v_i\rangle, \langle u_i\ \mathtt{output}\ g\_v_{i+1}\rangle,$
  $\langle u_i\ \mathtt{type\ delete}\rangle, \langle u_i\ \mathtt{data}\ g\_u_i\rangle$
  $\langle u_i\ \mathtt{source}\ s_1\rangle, \ldots, \langle u_i\ \mathtt{source}\ s_m\rangle,$
  $\langle u_i\ \mathtt{meta}\ m_i\rangle$, (metadata)$\}\}$

3. 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 Update. We have outlined an approach to the problem drawing on similar work in database archiving and copy-paste provenance in relational databases. 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. In particular, the metadata carried by our technique can use the PROV data model already developed by the W3C Provenance Interchange Working Group [2].

4. REFERENCES

[1] Peter Buneman, Adriane Chapman, and James Cheney. Provenance management in curated databases. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD '06, pages 539–550, New York, NY, USA, 2006. ACM.

[2] Luc Moreau and Paolo Missier. PROV data model. W3C Recommendation, April 2013. http://www.w3.org/TR/2013/REC-prov-dm-20130430/.