Dynamic Provenance for SPARQL Updates using Named Graphs

James Cheney and Harry Halpin

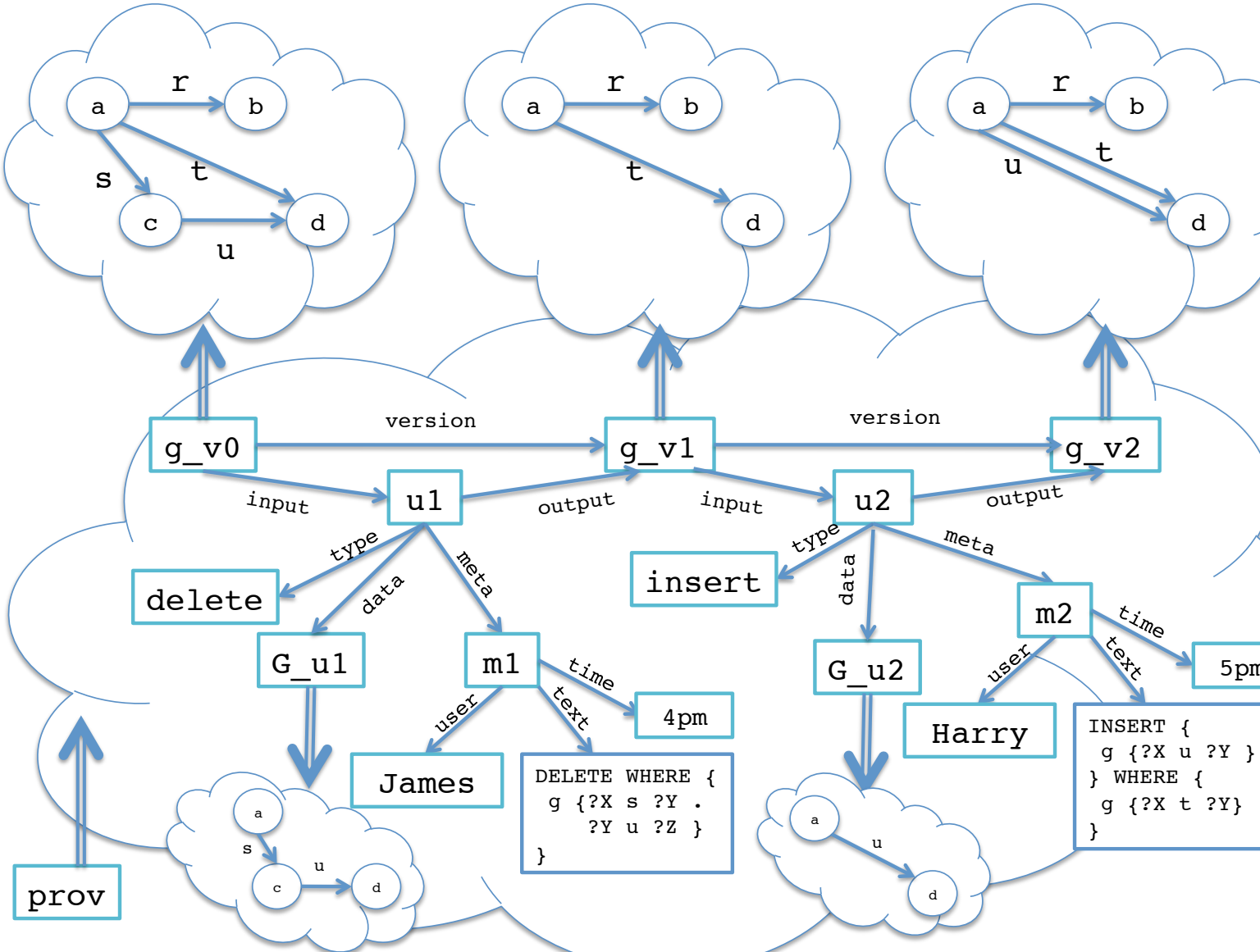
W3C/MIT (@harryhalpin harry@w3.org)

October 23, 2014

Presentation Outline

- 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.
- Our hypothesis is that a simple vocabulary, composed of insert, delete, and copy operations as introduced by Buneman et al., 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.

An Example


a
r
b
s
t
c
d
u
g_v0
version
g_v1
version
g_v2
input
u1
output
input
u2
output
type
data
meta
delete
insert
G_u1
m1
G_u2
m2
time
user
text
4pm
5pm
James
Harry
DELETE WHERE {
g {?X s ?Y .
?Y u ?Z }
}
INSERT {
g {?X u ?Y }
} WHERE {
g {?X t ?Y}
}
prov

Teminology

- Any graph that records the insertion and deletion of triples from a given graph is considered a *provenance graph* (*prov*) for the given graph. which keeps track of auxiliary named graphs.
- 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.

SPARQL Update Model

For queries, we consider a simple form of provenance which calculates a set of named graphs "consulted" by the query.

$$
\begin{aligned}
\mathcal{S}[\![C]\!]^{\mathcal{D}}_G &= \bigcup\{\mathrm{names}(\mu(C)) \mid \mu \in [\![C]\!]^{\mathcal{D}}_G\} \\
\mathcal{S}[\![P_1 \ . \ P_2]\!]^{\mathcal{D}}_G &= \mathcal{S}[\![P_1]\!]^{\mathcal{D}}_G \cup \mathcal{S}[\![P_2]\!]^{\mathcal{D}}_G \\
\mathcal{S}[\![P_1 \ \texttt{UNION} \ P_2]\!]^{\mathcal{D}}_G &= \mathcal{S}[\![P_1]\!]^{\mathcal{D}}_G \cup \mathcal{S}[\![P_2]\!]^{\mathcal{D}}_G \\
\mathcal{S}[\![P_1 \ \texttt{OPT} \ P_2]\!]^{\mathcal{D}}_G &= \mathcal{S}[\![P_1]\!]^{\mathcal{D}}_G \cup \mathcal{S}[\![P_2]\!]^{\mathcal{D}}_G \\
\mathcal{S}[\![P \ \texttt{FILTER} \ R]\!]^{\mathcal{D}}_G &= \mathcal{S}[\![P]\!]^{\mathcal{D}}_G \\
\mathcal{S}[\![\texttt{SELECT} \ ?\vec{X} \ \texttt{WHERE} \ P]\!]^{\mathcal{D}} &= \mathcal{S}[\![P]\!]^{\mathcal{D}} \\
\mathcal{S}[\![\texttt{CONSTRUCT} \ C \ \texttt{WHERE} \ P]\!]^{\mathcal{D}} &= \mathcal{S}[\![P]\!]^{\mathcal{D}}
\end{aligned}
$$

Create Graph

A graph creation CREATE g is translated to

$$
\begin{array}{l}
\texttt{CREATE } g; \\
\texttt{CREATE } g\_v_0; \\
\texttt{INSERT DATA } \{\texttt{GRAPH } prov \ \{ \\
\quad \langle g \ \texttt{version} \ g\_v_0 \rangle, \langle g \ \texttt{current} \ g\_v_0 \rangle, \\
\quad \langle u_1 \ \texttt{type create} \rangle, \langle u_1 \ \texttt{output} \ g\_v_0 \rangle, \\
\quad \langle u_1 \ \texttt{meta} \ m_i \rangle, (\text{metadata}) \\
\}\}
\end{array}
$$

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.

Clear Graph

A clear graph operation CLEAR g is handled as follows:

CLEAR g;
DELETE WHERE {GRAPH *prov* {$\langle g$ current $g\_v_i\rangle$}};
INSERT DATA {GRAPH *prov* {
$\langle g$ version $g\_v_{i+1}\rangle, \langle g$ current $g\_v_{i+1}\rangle,$
$\langle u_i$ 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)
}}

Clear Graph

A clear graph operation CLEAR g is handled as follows:

CLEAR g;
DELETE WHERE {GRAPH *prov* {$\langle g$ current $g\_v_i\rangle$}};
INSERT DATA {GRAPH *prov* {
$\langle g$ version $g\_v_{i+1}\rangle, \langle g$ current $g\_v_{i+1}\rangle,$
$\langle u_i$ 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)
}}

Load Graph

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

LOAD h INTO g;
DELETE WHERE {GRAPH $prov$ {$\langle g$ current $g\_v_i\rangle$}};
INSERT DATA {GRAPH $prov$ {
$\langle g$ version $g\_v_{i+1}\rangle, \langle g$ current $g\_v_{i+1}\rangle,$
$\langle u_i$ 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.

Insert Graph

An insertion $\texttt{INSERT}\ \{\texttt{GRAPH}\ g\ \{C\}\}\ \texttt{WHERE}\ P$ 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:

$$
\begin{array}{l}
\texttt{CREATE}\ g\_u_i; \\
\texttt{INSERT}\ \{\texttt{GRAPH}\ g\_u_i\ \{C\}\}\ \texttt{WHERE}\ P; \\
\texttt{INSERT}\ \{\texttt{GRAPH}\ g\ \{C\}\}\ \texttt{WHERE}\ P; \\
\texttt{CREATE}\ g\_v_{i+1}; \\
\texttt{LOAD}\ g\ \texttt{INTO}\ g\_v_{i+1}; \\
\texttt{DELETE DATA}\ \{\texttt{GRAPH}\ \mathit{prov}\ \{\langle g\ \texttt{current}\ g\_v_i\rangle\}\}; \\
\texttt{INSERT DATA}\ \{\texttt{GRAPH}\ \mathit{prov}\ \{ \\
\quad \langle g\ \texttt{version}\ g\_v_{i+1}\rangle, \langle g\ \texttt{current}\ g\_v_{i+1}\rangle, \\
\quad \langle u_i\ \texttt{input}\ g\_v_i\rangle, \langle u_i\ \texttt{output}\ g\_v_{i+1}\rangle, \\
\quad \langle u_i\ \texttt{type insert}\rangle, \langle u_i\ \texttt{data}\ g\_u_i\rangle \\
\quad \langle u_i\ \texttt{source}\ s_1\rangle, \ldots, \langle u_i\ \texttt{source}\ s_m\rangle,
\end{array}
$$

Delete Graph

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

CREATE g_u_i;
INSERT $\{$GRAPH g_u_i $\{C\}\}$ WHERE P;
DELETE $\{$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 delete$\rangle, \langle u_i$ 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)$\}\}$

Dynamic Provenance Vocabulary

Name	Description
upd:insert	Triples inserted into a graph.
upd:delete	Triples deleted from a graph.
upd:load	Triples copied from another graph.
upd:clear	All triples in graph deleted
upd:create	New graph initialized.
upd:drop	A graph is deleted.
upd:input	Linked to graph used as input in update operation.
upd:output	Link to graph of output of the operation.
upd:data	Changed data in insert/delete operation.
upd:version	Link between two versions of a graph.
upd:type	Type of update operation.
upd:current	Link to most current state of graph.
upd:source	Any other graph that was consulted by the update.
upd:meta	Link to any metadata about the graph.

Lightweight Dynamic Provenance Vocabulary

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. In particular, the metadata carried by our technique can use the PROV data model already developed by the W3C Provenance Interchange Working Group