Application of Named Graphs Towards Custom Provenance Views

Tara Gibson, Karen Schuchardt, Eric Stephan
Pacific Northwest National Laboratory

Abstract

Provenance capture as applied to execution oriented and interactive workflows is designed to record minute detail needed to support a "modify and restart" paradigm as well as re-execution of past workflows. In our experience, provenance also plays an important role in human-centered verification, results tracking, and knowledge sharing. However, the amount of information recorded by provenance capture mechanisms generally obfuscates the conceptual view of events. There is a need for a flexible means to create and dynamically control user oriented views over the detailed provenance record. In this paper, we present a design which leverages named graphs and extensions to the SPARQL query language to create and manage views as a server-side function, simplifying user presentation of provenance data.

1. Introduction

Today's scientific workflow tools are built to accurately represent the execution of a diverse set of scientific activities. During a single workflow execution, hundreds to thousands of events may be triggered which add no intrinsic value to human-centered verification, results tracking, and knowledge sharing. Although the detailed workflow record is essential for reproducibility and detailed debugging, provenance must also be raised to a conceptual level that represents the scientist’s methodology.

Activity tracking systems, which dynamically record user-driven process execution, share similar problems capturing and presenting provenance at a suitable level for user presentation. While it is possible to customize the level of detail captured by the recording mechanism, it is simply not practical to constantly change the code to mirror the user’s conceptual perspective which can vary significantly across a diverse user base as well as over time.

Workflow provenance is naturally represented by graphs using standards such as Resource Description Framework (RDF). RDF provides a very flexible model for describing any data and is adaptable as models change over time. This flexibility, while not necessarily required to describe execution provenance, enables the integration of workflow provenance within a larger context. This flexibility is the reason that RDF is being adopted as a representation for many types of data with a dynamic or open vocabulary (e.g. biology, social networks) that have similar needs for user views. Similar to how views were applied to the relational model [1], views are needed as a general capability for the graph data model.

In this paper we propose a filtering technique that extends the SPARQL query language to help avoid information overload and the pitfalls of constantly rewriting custom recording or viewing mechanisms. Our approach leverages extensions to RDF Named Graphs (NG). With these extensions, client applications can render provenance in a way that is meaningful to users and enable users themselves to control the views. While we apply this approach specifically to workflow provenance, it is equally applicable to other graph-based data.

2. Related Work

The workflow community has long recognized data overload or noisiness as a problem. Altintas et al [2] proposes controls which screen the level of detail that is captured. Missier et al [3] propose a technique to control the level of detail collected by workflow designers and propose support for filtered views through query mechanisms that disregard certain lineage chains and/or exclude specific processors from the lineage graph. Cohen-Boulakia proposes views based on a grouping mechanism [4], but this grouping mechanism appears to be dependent on workflow design specification and therefore tied specifically to views of workflows. Similarly, research in the semantic web has pointed out the need for view management in a different context, namely creation of views across a wide spectrum of distributed data – the semantic web itself. [1]

[1] http://www.openarchives.org/ore/1.0/primer Oct. 2008

NGs are an important concept within the RDF community. Conceptually an RDF database can represent large, complex graphs. NGs provide a means to aggregate and assert custom statements relating to sets of triples within the database. They have been proposed as a means to share sub-graphs in the semantic web, as a basis for establishing trust, and generally as a means to support views [5]. Recent extensions to NGs support aggregation of a particular set of statements as well as NGs that are defined by queries [6].

In the context of workflow provenance views, NGs can also serve as a dynamic filtering mechanism for complex graphs. While it is possible to create a NG that provides a static view, that graph may become outdated as new information is added. Further, some of the view may contain repeating patterns that should always be aggregated and thus require an element of dynamic rule evaluation. Our proposal provides for this type of dynamic view creation by applying NG aggregations to query results.

3. Motivating Example

To show the utility of filtered views we consider its use in a simulation workbench. The workbench utilizes a workflow execution tool to schedule and execute simulations as well as stage data to and from compute servers. In addition, it consists of an activity management application that assists with the interactive, complex simulation setup and analysis of results. Provenance is captured for both the activity environment and the workflow environment. Our objective is to use the provenance record both as a workspace for conducting numerical studies and as a means to answer questions about results derivation.

As the numerical studies are conducted, the simulation workbench records the workflow by capturing the process and data relationships in the Open Provenance Model (OPM) [7] model. To review the results, users need to see a high level summary of their process execution history without the clutter generated by presenting all of the data artifacts and parameters. Figure 1 shows a small graph fragment before and after aggregation is applied where the aggregation groups output data with the process that created them. In practice, the number of files far exceeds the number of processes such that this filtering accomplishes significant view reduction.

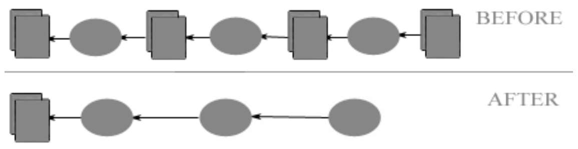

Figure 1. Aggregation applied to an OPM modeled graph for a process view of the model.

Within the process view, there are usually data translation steps that add little in the way of meaningful detail. They typically can be viewed as a pre-processing activity to a more conceptually meaningful end-user activity. Pairing these can greatly reduce graph clutter.

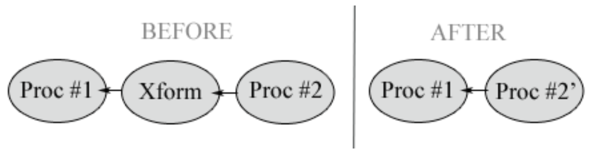

Figure 2. Aggregation of preprocessing steps to reduce intermediate results.

Similarly, post-processing activities can be considered as uninteresting detail. As shown in the figure below, post-processing generates data which together with the original source data can be applied to analysis steps in a looping manner. The graph for this is very complex. However, if the post-processing step is aggregated with the data generation activity, the information presented is clear.

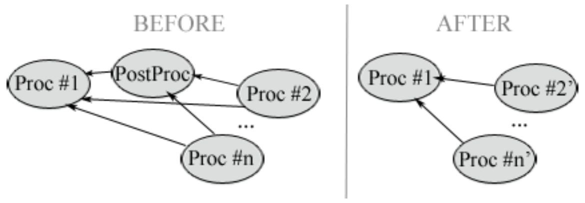

Figure 3. Aggregation of post processing steps.

Finally, in the general case, aggregation can be used to group an arbitrary set of specific processes as in the Figure 4. This type of aggregation also supports truncation which is useful in a number of contexts. It can be used to simply remove data down an execution path based on the type of process or it can be used to support end-user annotation and hiding of uninteresting chains of investigation.

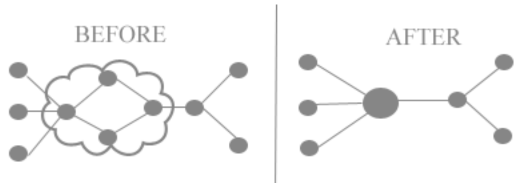

Figure 4. General user-defined aggregation.

Figure 5a) provides an example of a larger graph that contains all of the patterns described in Figures 1 through 4. Figure 5b) shows the graph after reduction to a process-to-process model. Figure 5c) shows the graph after reducing the patterns in Figures 2 through 4.

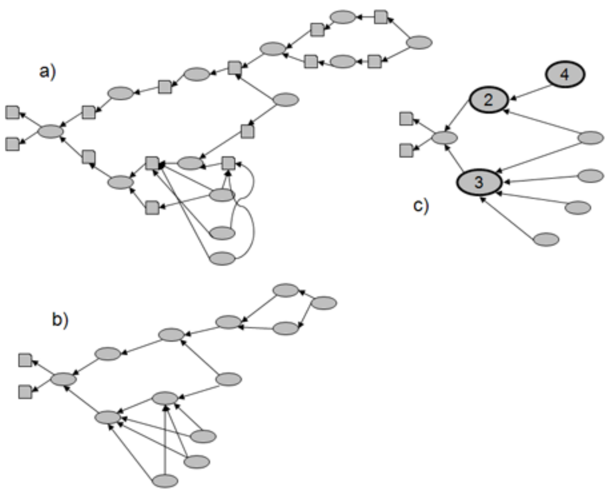

Figure 5. Example demonstrating multiple views applied to achieve significant complexity reduction.

The aggregated nodes are enlarged and numbered according to the type of aggregation performed where the numbers tie back to the previous figures. Clearly the visual clutter is greatly reduced. The reduction of clutter and hiding the uninteresting detail results in a graph can meaningfully convey more information to a user. Our specific contribution is the use and extension of NGs and the introduction of the APPLY keyword to the SPARQL query language, to support complex and nested view reduction operations.

4. Design

A powerful NG syntax, described by Schenk et al, supports both explicit triples and query notations [5]. The following example illustrates a basic NG with the triple structure of RDF and the use of explicit references using Trig[2] notation. In this example, RDF statements are listed in curly brackets, prefaced by the name of the graph. Due to space constraints, we will use abbreviated URIs without namespaces.

```
:DBLP {
    :proc2 dc:title :"Proc #2".
    :proc2 :from :xform.
    :xform dc:title "Xform"
}
```

Figure 6. Named Graph with static triples.

[2] http://www4.wiwiss.fu-berlin.de/bizer/TriG/

An example NG using dynamic query notation is shown in Figure 7. The view definition is included in statements of the form: ng:definedBy <query> where <query> is a literal containing a CONSTRUCT query. Such a statement is called a view definition statement. A NG may contain a number of these view definition statements, all of which contribute to the dynamic NG [6]. In order to meet the needs of custom user views, we also require the extension of SPARQL-style variables within the view definition. The variables (in **bold**) should match those created in the query CONSTRUCT.

```
:proc1Agg {
     :proc1Agg :targetURI ?proc.
     :proc1Agg :type ?type.
     :proc1Agg :title ?title.
     :proc1Agg :someProp1 "someVal".
    : proc1Agg ng:definedBy
        "CONSTRUCT {?proc ?pred ?obj.
                    ?proc :title ?title.
                    ?proc :type ?type.
                    ?data ?pred ?obj}
        WHERE { ?proc :type :Process.
                ?data :generatedBy ?proc.
                ?data :type :Data.
        }".
}
```

Figure 7. Dynamic Named Graph

The above notations can describe some fairly complex and dynamic NGs. In particular, the query demonstrates how data artifacts in Figure 1 can be filtered out of the workflow graph. By adding query extensions to this NG notation, we can support the reductions shown in Figure 5. Our SPARQL grammar extension provides methods to replace and filter results by defining a NG model where the user view is specified with at least a targetURI, type, and title. The new keyword 'APPLY' tells the query interface which views should be applied to the query result. Based on the specified views, each NG is aggregated into a logical, user-defined node. This new node is then created based on the properties associated with the NG definition.

An example that demonstrates Figures 1 & 2 will clarify the design. The initial query result contains all of the process and data resources from a given workflow. Within this graph are the processes named *proc#1*, *Xform*, and *proc#2*. The user wants to apply two NGs, the first of which filters out data artifacts by combining them with their generating process as seen in Figure 1 and the second contains the resources *Xform* and *proc#2* as seen in Figure 2. The NG definition in Figure 7 depicts the view shown in Figure 1, and the definition in Figure 8 depicts the view for Figure 2.

```
:proc2Agg {
     :proc2Agg :targetURI :proc2'.
     :proc2Agg :title "proc#2`".
     :proc2Agg :type :Process.
     :proc2Agg :someProp1 "someVal".
    : proc2Agg g:definedBy
        "CONSTRUCT {?proc2 ?pred ?obj.
                              ?xform ?pred ?obj}
        WHERE { ?proc2 :type :Process2.
                       ?proc2 :from ?xform.
                       ?proc2 ?pred ?obj.
                       ?xform ?pred ?obj.
                       ?xform :type :XForm
        }".
}
```

Figure 8. A Dynamic Named Graph for Figure 2.

Given these NGs, the query would then be formulated with the new APPLY keyword (Figure 9). The APPLY keyword is appended to the end of the standard SPARQL query, and it should be used after all other standard SPARQL keywords. The keyword should be followed by a comma delimited list of NGs to be applied to the result set. There may also be a second modifier, 'AS', to control the output format of the query result. The response format may be one of the several standard RDF serializations (RDF/XML, N3…), or a standardized graph format such as GraphML or GXL.

```
CONSTRUCT {?sub ?pred ?obj}
WHERE {?sub :inWorkflow :WorkflowX.
            ?sub ?pred ?obj
}
APPLY :proc1Agg, :proc2Agg. AS GraphML
```

Figure 9. Query using the named graphs for filtering.

The query in Figure 9 executes as follows. First the primary query (everything except APPLY) is performed and the results are cached. Next, the *:proc1Agg* NG is applied to the full query result graph producing a reduced graph where each process is combined with its output data. As defined in Figure 7, the targetURI in this NG definition is linked to each process URI. Therefore, an aggregated node will be created for each process and given the type and title from that process node. Finally, the *:proc2Agg* NG is applied to the reduced graph which removes the contents *Xform* and *Proc#2* and replaces them with a single node titled *Proc#2*' as defined in Figure 8. As this example demonstrates, it is important to apply multiple views to achieve the desired results. In this case, the first view provides the reduction described in Figure 1. The second view provides the reduction described in Figure 2. The order of evaluation must be carefully considered by the issuer of the query since the data within the graph changes after each application of a view. A well defined naming mechanism is necessary to identify which components (NGs) should be used to create these layered views.

When a query with a NG filter returns aggregated nodes in the graph, it may be useful for client tools to view the detail of the aggregate node. This 'drill down' functionality is inherently provided by SPARQL and NGs. As defined in http://www.w3.org/TR/rdf-sparql-query/#specDataset a SPARQL query may use the 'FROM NAMED' clause to specify the NG to be used for querying against (rather than the entire RDF graph.) This allows for queries to retrieve all or a subset of the values within the NG.

5. Discussion

The design as presented leverages NGs to create and manage views as a server-side function, simplifying user presentation of provenance data. By including this server-side functionality through SPARQL, users already familiar with the query language can use this functionality in a standardized way. Making this a server-side capability opens an opportunity for optimization which we have yet to explore. Scalability is another concern. In a basic implementation, all queries and applications of NGs would be performed dynamically. This approach will work as long as the initial query result set is not too large. For large result sets, our expectation is that the performance will decrease. However, it is possible for more sophisticated implementations to create persistent views and update these dynamically as required.

It is our intention to implement the described functionality in the Sesame framework. This includes functions to transitively search or 'walk' the graph. For OPM and likely other data models, it is necessary to be able to traverse the graph using multiple predicates and specify stop conditions. We also intend to implement the APPLY keyword. We will evaluate both the performance and the suitability of this approach when applied to a variety of use cases focused on provenance graphs generated by both workflow systems and activity tracking systems.

Acknowledgments

This research is supported by the U. S. Department of Energy's Office of Science under the Scientific Discovery through Advanced Computing (SciDAC) program and through the Laboratory Directed Research and Development Program at the Pacific Northwest National Laboratory operated by Battelle for the U.S. Department of Energy under contract DE-AC05-76Rl0 1839.

References

1. Codd, E. (1982). "Relational database: a practical foundation for productivity." Communications of the ACM 25(2): 109-117.
2. Altintas, I., Barney, O., & Jaeger-Frank, E. (2006). Provenance collection support in the Kepler Scientific Workflow System, In *International Provenance and Annotation Workshop* (IPAW), LNCS, Provenance and Annotation of Data, 4145: 118-132, 2006.
3. Paolo Missier, Khalid Belhajjame, Jun Zhao, Carole Goble, *Data lineage model for Taverna workflows with lightweight annotation requirements*, IPAW'08, Springer LNCS series, vol. 5272/2008, Salt Lake City, Utah, June 2008.
4. Cohen-Boulakia, S., Biton, O., Cohen, S., and Davidson, S. 2008. Addressing the provenance challenge using ZOOM. *Concurr. Comput. : Pract. Exper.* 20, 5 (Apr. 2008), 497-506.
5. Carroll, J.J., Bizer C., Hayes P., Stickler P. “Named Graphs, Provenance and Trust.” In *Proceedings of the 14th international conference on World Wide Web*. 2005, Chiba, Japan. New York; ACM, pp. 613-622.
6. Schenk, S. and Staab, S. 2008. Networked graphs: a declarative mechanism for SPARQL rules, SPARQL views and RDF data integration on the web. *In Proceeding of the 17th international Conference on World Wide Web* (Beijing, China, April 21 – 25, 2008). WWW ’08. ACM, New York, NY, 585-594
7. Moreau L, J Freire, J Myers, J Futrelle, and PR Paulson. 2007. "The Open Provenance Model." Presented by Luc Moreau at Workshop on Principles of Provenance, Edinburgh, Scotland on November 20, 2007