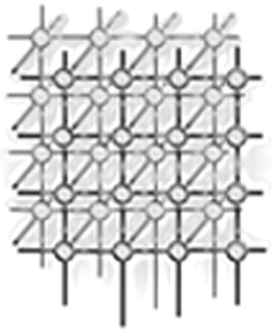
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
*Concurrency Computat.: Pract. Exper.* 2007; **00**:1–10 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*

# Query capabilities of the Karma provenance framework



Yogesh L. Simmhan*,†, Beth Plale and Dennis Gannon

*Computer Science Department, Indiana University, Bloomington IN 47405, USA.*

**SUMMARY**

**Provenance metadata in e-Science captures the derivation history of data products generated from scientific workflows. Provenance forms a glue linking workflow execution with associated data products, and finds use in determining the quality of derived data, tracking resource usage, and for verifying and validating scientific experiments. In this article, we discuss the scope of provenance collected in the Karma provenance framework used in the LEAD Cyberinfrastructure project, distinguishing provenance metadata from generic annotations. We further describe our approaches to querying for different forms of provenance in Karma in the context of queries in the first provenance challenge. We use an incremental, building-block method to construct provenance queries based on the fundamental querying capabilities provided by the Karma service centered on the provenance data model. This has the advantage of keeping the Karma service generic and simple, and yet supports a wide range of queries. Karma successfully answers all but one challenge query. Copyright © 2007 John Wiley & Sons, Ltd.**

KEY WORDS: data and process provenance; scientific workflows; provenance queries

## 1. INTRODUCTION

Provenance in e-Science [22, 2] is a form of metadata capturing the derivation history of data products generated from executing scientific tasks, often modeled as workflows. Provenance forms the glue linking workflow executions with associated data products, and finds use in determining the quality of derived data, tracking resource usage, verifying and validating scientific experiments, and for information discovery and integration through querying and mining.

The Karma provenance framework [23] is used in the LEAD Cyberinfrastructure [7, 11] project to collect and query over provenance generated from meso-scale weather forecasting workflows. The Karma system was one of the entries in the first provenance challenge[17] and this article describes our

*Correspondence to: Yogesh L. Simmhan, Computer Science Department, Indiana University, 150 S. Woodlawn Ave., Bloomington IN 47405, USA.
†E-mail: ysimmhan@cs.indiana.edu
Contract/grant sponsor: National Science Foundation; contract/grant number: ATM-0331480


*Received 29 November 2006*
*Revised 24 April 2007*
*Accepted 1 May 2007*

experiences in addressing the challenge problems. Karma places a greater emphasis on the collection of provenance from workflow executions as compared to providing rich querying capability. The provenance challenge was an opportunity to test different approaches for querying for provenance in Karma, which provided an insight on future enhancements to the query interface.

Karma takes a conservative view of the scope of provenance metadata, limiting it largely to the causal chain of events that lead to the creation of data products through the execution of processes. Generic annotations are considered beyond the purview of a provenance system. There are several general purpose metadata catalogs available for various resources on the Grid [6, 25, 24] and the provenance system can play a pivotal role in the integration of information from these information services – without being tasked with storing, managing, and querying generic annotations. We had a chance to pit this philosophy against the challenge queries, several of which deal with queries over annotations, and found our system to answer all but one of the challenge queries.

The rest of this article is organized as follows: in Section 2, we briefly describe the Karma provenance framework and its data model, followed by a discussion of the query capabilities of Karma and its application to the challenge problem in Section 3, and we conclude in Section 4 with some comparisons with other provenance systems in the challenge and our future work.

## 2. THE KARMA PROVENANCE FRAMEWORK

The Karma provenance framework [23] is used to collect and query over provenance on data products derived from scientific workflows executing in a service oriented architecture. Scientific experiments in the LEAD project are designed as workflows composed out of web services. These services encapsulate specific scientific applications, taking input data and parameters and generating output data, and are usually command-line applications wrapped using our custom Service Factory Toolkit that generates web services wrappers for arbitrary scientific tasks [12, 11]. Services used in workflows may themselves be workflows, allowing for hierarchical workflow composition . LEAD uses a variation of Business Process Execution Language (BPEL) as the workflow description language and a centralized workflow engine to orchestrate service invocations on the LEAD Grid [26].

**Provenance Activities.** Provenance collection is a continuous process during the lifetime of a workflow. Pieces of the provenance is collected over time from different workflow components in the form of *provenance activities*, and integrated into the complete provenance model by a provenance service. Clients and services in the workflow are instrumented to generate provenance activities [23]. Activities identify the boundaries of execution of the workflow and the data dependencies for the service. They are generated by the workflow engine and the service at the beginning and ending of a service invocation, and by the service when it produces or consumes data.

All services and data products are uniquely identified by a global ID. The granularity of data products is arbitrary – and can be any identifiable data resource that can be mapped to from the global ID. A service invocation is a complex key comprising of the service and client states in the workflow at the point of invocation. This state is captured by the *entity ID* that identifies the workflow the service is part of, the service instance, the logical time in the workflow lifecycle when the service is invoked, and the unique node the service maps to in the workflow graph. The client to the service, which is usually the workflow engine, has a similar entity ID defined for it. Combining the entity IDs for the client and service makes is possibly to uniquely identify the invocation in time and space. These entity IDs and

```
<serviceInvoked xmlns="http://lead.extreme.indiana.edu/namespaces/2006/06/workflow_tracking">
 <notificationSource serviceID="urn:qname:http://www.extreme.indiana.edu/karma/challenge06:ConvertService"
  workflowID="tag:gpel.leadproject.org,2006:69B/ProvenanceChallengeBrainWorkflow17/instance1"
  workflowNodeID="ConvertService_4" workflowTimestep="36"/>
 <timestamp>2006-09-10T23:56:28.677Z</timestamp> <description>Convert Service was Invoked</description>
 <request><header>...</header><body>...</body></request>
 <initiator serviceID="tag:gpel.leadproject.org,2006:69B/ProvenanceChallengeBrainWorkflow17/instance1" />
</serviceInvoked>
```

(a)

```
<dataProduced xmlns="http://lead.extreme.indiana.edu/namespaces/2006/06/workflow_tracking">
 <notificationSource serviceID="urn:qname:http://www.extreme.indiana.edu/karma/challenge06:ConvertService"
  workflowID="tag:gpel.leadproject.org,2006:69B/ProvenanceChallengeBrainWorkflow17/instance1"
  workflowNodeID="ConvertService_4" workflowTimestep="36" />
 <timestamp>2006-09-10T23:56:32.324Z</timestamp>
 <dataProduct>  <id>lead:uuid:1157946992-atlas-x.gif</id>
  <location>gsiftp://tyr1.cs.indiana.edu/tmp/20060910235628_Convert/outputData/atlas-x.gif</location>
  <timestamp>2006-09-10T23:56:32.324Z</timestamp>  </dataProduct>
</dataProduced>
```

(b)

```
<dataProvenance xmlns="http://extreme.indiana.edu/namespaces/2006/08/karma/xsd"
  xmlns:wft="http://lead.extreme.indiana.edu/namespaces/2006/06/workflow_tracking"
  dataProductID="lead:uuid:1157946992-atlas-x.gif" timestamp="2006-09-10T23:56:32.324Z"
  dataProductLocation="gsiftp://tyr1.cs.indiana.edu/tmp/20060910235628_Convert/outputData/atlas-x.gif">
 <producedBy status="InvokingService ServiceInvoked InvokingServiceSucceeded
   SendingResult ReceivedResult SendingResponseSucceeded"
  requestReceiveTime="2006-09-10T23:56:28.677Z" responseSendTime="2006-09-10T23:56:32.397Z">
  <invoker wft:serviceID="tag:gpel.leadproject.org,2006:69B/ProvenanceChallengeBrainWorkflow17/instance1" />
  <invokee wft:serviceID="urn:qname:http://www.extreme.indiana.edu/karma/challenge06:ConvertService"
   wft:workflowID="tag:gpel.leadproject.org,2006:69B/ProvenanceChallengeBrainWorkflow17/instance1"
   wft:workflowNodeID="ConvertService_4" wft:workflowTimestep="36" isWorkflow="false" />
 </producedBy>
 <usingData dataProductID="lead:uuid:1157946967-atlas-x.pgm" timestamp="2006-09-10T23:56:32.180Z"
  dataProductLocation="gsiftp://tyr7.cs.indiana.edu/tmp/20060910235600_Slicer/outputData/atlas-x.pgm" />
</dataProvenance>
```

(c)

Figure 1. (a) A ServiceInvoked activity generated by a service upon invocation. (b) DataProduced activity generated by a service when it creates a new data product. (c) The data provenance for a data product.

data product IDs are present in the activities that are generated, along with relevant metadata such as timestamps and data product locations.

The provenance activities adhere to a well-defined XML schema. Activities can additionally contain generic XML metadata as annotations. This is provided for convenience. Two sample activities generated by an invoked service are shown in Figure 1(a) and (b). While the ServiceInvoked activity is generated at the moment the service is called, the DataProduced activity is generated whenever a data product is created by the service during the invocation. The DataProvenance model shown in Figure 1(c) is built from these and other activities, and described in the next section.

The activities form an event driven approach to collecting provenance. The data products exchanged between service invocations establishes a data centric causality trail. In LEAD, the workflow engine invoking the services is instrumented to generate client-side provenance activities, while the service wrappers generated for the scientific tasks are automatically instrumented to generate service and data provenance activities. Karma supports both a notification based model to asynchronously listen to these activities and also provides a direct web service API to submit the provenance activities synchronously.

**Provenance Data Model.** Karma exports two primary forms of provenance that are constructed from the activities: *data provenance* and *process provenance*, with variations of each [23]. Process provenance describes the invocation of a service, along with the inputs and outputs of data products to it. Data provenance describes the derivation of a data product, including the service invocation that created this data and the inputs to the invocation. Both these types of provenance are represented as XML documents that is defined by the provenance model. Data products in the model have the unique data product ID, timestamp of creation or usage, and the location of the data product (replica). Service invocations are represented by the complex key formed from the client and service entity IDs. The clients and services are also known as the invokers and invokees, making an invocation a pair of invoker and invokee.

A typical data provenance for the data product with ID lead:uuid:1157946992-atlas-x.gif is shown in Figure 1(c). This shows the data product being created on 10th September 2006 at 11:56PM UTC at the tyr1 host by the ConvertService running as part of the ProvenanceChallengeBrainWorkflow17 workflow and invoked by the workflow engine that acts as the client, passing a data product having ID lead:uuid:1157946967-atlas-x.pgm as the input.

In addition to the *immediate provenance*, data provenance can be recursively tracked beyond the immediate service invocation creating it. Since other data products were used as inputs to the invocation, the immediate provenance for those ancestral data products can be attached to the data provenance to recursively render a *deep or recursive provenance* tree going back in time. A corollary of this data provenance is a *usage trail* for the data. This returns the service invocations that use a data product and can be used to go forward in time along the data provenance graph.

Similarly, variations of process provenance exists. The workflow trace describes the process provenance for all services invocation that were part of a workflow run. This forms a directed graph where service invocations are nodes and data products are edges. Since workflows themselves are abstracted as services, this method can be recursively applied to build a hierarchical provenance graph for service and workflow invocations of arbitrary depth that were executed across organizational boundaries. This conveniently presents different levels of abstraction to the user to retrieve process provenance – at the fine-grained level of a service invocation or at the coarser granularity of a workflow. Since workflows can be hierarchically composed, this allows abstraction of process provenance to arbitrary coarseness based on the workflow design.

All of these provenance models can be constructed from the same set of provenance activities generated from the workflow.

**Backend Database Model.** Provenance activities that arrive as XML documents at the provenance service are decomposed into their essential attributes, and stored in a relational database. Shredding the XML into a relational model allows convenient querying over the provenance and allows it to later be disseminated in any representation. Hence the ability to construct different forms of provenance like process provenance or recursive data provenance from the same set of activities.

The relational schema, outlined in Figure 2, stores the various facets of provenance: services and clients, known as entities; details about services (which are potentially workflows); invocations of the services by clients like the workflow engine; data products and their relationship to entities that consume and produce them; and finally the raw provenance activities, called notifications, for provenance about the provenance data itself. In addition to standard fields such as timestamp and unique IDs, the data product, the service, and the notification tables allow generic XML metadata to

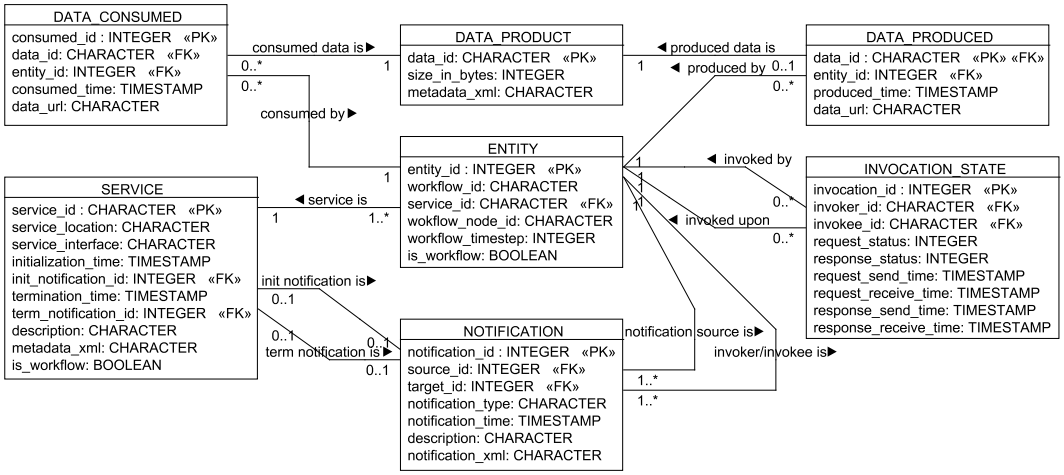



Figure 2. Relational Data Model for Provenance Activities. The Entity table captures information about the source or target of an invocation, the Invocation table relates the source and target of an invocation with the status of the invocation, and the Service table stores details about individual service entities. Between them, they record the *process provenance*. Data Product, Data Produced, and Data Consumed tables maintain details about data created and used by each entity, and record the *data provenance*. The raw provenance activities and annotations are available through the Notification table.

be recorded for convenience and these can be retrieved as part of provenance for post processing. However, only free-text querying is possible upon them and no syntactic structure is pre-supposed.

The information in these tables can be queried and combined to build different forms of provenance. The basic queries supported by the provenance service API are based on the unique IDs of the data products, services, workflows, and invocations. These are internally translated to SQL queries on the different tables and the relevant tuples extracted to populate the provenance data model. For example, a query for the data provenance of data product with ID lead:uuid:1157946992-atlas-x.gif would create a document similar to Figure 1(c), generated by joining information from the data product and data produced tables to identify the data product, the entity and invocation state tables to locate the service and invocation that created this data product, and the data consumed table for invocations that have used this data product. Karma uses the MySQL relational database as the backend implementation.

## 3. QUERY CAPABILITIES

The provenance challenge gives a unique chance to showcase the querying capabilities of Karma. The challenge workflow, described more fully in the introduction to this issue [17], is composed as a BPEL workflow [12, 26] using the XBaya Workflow Composer and executed using the BPEL workflow engine on a local cluster. Each process in the workflow is a web service wrapping a shell

Table I. Approaches used to perform challenge queries. Those answered using the service API are marked API; those requiring post-processing by client are under Client; those answered using direct SQL query on database are marked SQL; while the one that was not answered is marked Unsupported.

| Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 | Q9 |
|---|---|---|---|---|---|---|---|---|
| API | Client | API | SQL | SQL | SQL | Client | SQL | Unsupported |

script using the Service Factory Toolkit, and the shell script invokes the corresponding command-line application it is mapped to, such as AlignWarp or Softmean. The services run on different hosts, and all services and data products are assigned unique IDs when created. The web service wrapper automatically registers the services with the Resource Catalog service registry, stages files between hosts using third-party file transfers, registers the data product replicas with a naming service, and generates provenance activities as asynchronous notifications published to a notification broker. The Karma provenance service subscribes to all provenance activities from the workflow run, stores them in the relational database, and makes them available for querying.

The queries defined for the challenge [17] can be grouped into those based purely on the structure of provenance and those requiring the additional inspection of annotations. *Provenance structure* queries pertain to information such as *"which process created a data product"*, *"when was it created"*, *"what were the input data products to a process"*, and these may be applied recursively to the data derivation chain. *Annotation queries* are over additional metadata submitted beyond the scope of provenance that can be generic in nature, such as parameters and attributes like center=UChicago describing the deriving process or the data products. In the challenge queries, Q1, Q2, Q3, and Q8 are provenance structure queries, while Q4, Q5, Q6, Q7, and Q9 rely on annotations also.

Three different techniques are used to answer the challenge queries. Some of the queries on the provenance structure can be directly answered by the provenance query API exposed by the Karma service. Other provenance structure queries require the client to do incremental queries by processing the results of a service API query and performing additional service queries based on the results. Finally, queries that involve annotations are directly written as SQL queries over the backend relational database. The approach used to answer each of the challenge queries is given in Table I. In the ensuing sections, we provide one example for each of these approaches – API, Client, and SQL – as applied to a challenge query. Other challenge queries that follow the same querying approach are not discussed in the interests of brevity. All workflows, services, queries performed, and their results are available online at the provenance challenge website [9].

*Directly Answered Queries (API):* Queries Q1 and Q3 can be directly answered using the Karma service API. The service API supports several methods to retrieve data and process provenance, workflow trace, and data usage metadata using the global IDs of workflows, service invocations, and data products. The service API is intentionally simple, presenting the methods `getProcessProvenance`, `getWorkflowTrace`, `getDataProvenance`, `getRecursiveDataProvenance`, and `getDataUsage`. The process provenance and workflow trace both return process provenance metadata given the invocation ID for a service or workflow invocation. While the latter returns the process provenance for all services that were part of the workflow, the former returns this information only for a single service invocation. The workflow trace also optionally does a recursive retrieval of the process provenance of other child workflows invoked

as part of this workflow, and the depth of the recursion is configurable. The data provenance can be retrieved for a given data product ID and the recursive variant works backwards in time, fetching the data provenance for the inputs to the process generating this data product and so on. Again, the depth of recursion is parameterized. Finally, the data usage for a data product ID works forward in time, locating the processes that use a certain data product after it is created.

Query Q1 is to *find the process that led to Atlas X Graphic*. This requires us to generate the complete data provenance for the Atlas X Graphic file recursively. This is readily achieved by using the `getRecursiveDataProvenance` method with the depth of recursion set to unlimited. This works backwards from the Atlas X Graphic file and returns all processes and their input and output data products that went to create the Atlas X Graphic file. A client-side Java library is provided to invoke the provenance web service and the returned provenance metadata can be accessed through XML Bean Java objects for convenience. The library call to get the recursive data provenance for the Atlas X Graphic file with data product ID lead:uuid:1157946992-atlas-x.gif is given below. The $-1$ parameter signifies an unlimited recursion depth.

```
RecursiveDataProvenanceType dataProvenance = karmaStub.
 getRecursiveDataProvenance("lead:uuid:1157946992-atlas-x.gif", -1);
```

Similarly, query Q3 to get the data provenance for the Atlas X Graphic file restricted to 3 levels back can be answered by the same method above, replacing the $-1$ for recursion depth by 3.

*Queries Requiring Client-side processing (Client):* In order to answer more complex queries that cannot be retrieved through a single service API call, clients use an incremental building-block approach to fetch the necessary provenance information from the Karma service. Queries Q2 and Q7 in the challenge fall under this category. Query Q2 is similar to Q3 discussed previously and retrieves the recursive data provenance for the Atlas X Graphic file. However, instead of limiting the depth of recursive data provenance, we stop the recursion when a particular process is encountered in the data provenance. For query Q2, this process is Softmean.

Limiting data provenance recursion based on the process is not directly supported by the service API. So we use the `getDataProvenance` API to get the immediate data provenance for the Atlas X Graphic file, and for each input to the deriving process, we repeatedly call `getDataProvenance`. This allows us to move backwards in the provenance tree until the Softmean process is reached. The pseudo-code for the client looks as follows:

```
DataProvenance[] dataProvenance = RecursiveDataProvenanceUntil(
 'lead:uuid:1157946992-atlas-x.gif','urn:qname:...:SoftmeanService')

function RecursiveDataProvenanceUntil(DataProductID dataProductID,
                                      URI processID) :: DataProvenance[]
1. $resultList[] = {}
2. $dataList[] = {dataProductID}
3. while ($dataList.size != 0)
   a. $dataProvenance = karmaStub.getDataProvenance($dataList[0])
   b. $resultList.add($dataProvenance)
   c. $dataList.remove(0)
   d. if ($dataProvenance.producedBy == processID) break
   e. foreach ($inputData in $dataProvenance.usingData) do
      i. $dataList.add($inputData)
4. return $resultList;
```

Query Q7 to find the differences between two workflow runs with slightly different services also requires post-processing by the client. For this, the workflow trace for both workflow executions are retrieved and the two graphs recursively compared by the client to find the differing processes. Standard algorithms exist to do a `diff` on two graphs (which the workflow traces are) and one such algorithm is used to solve this. Differences that can be identified include variations in processes, differences in input and output data created, changes in invocation and creation times, and other details available as part of the process provenance data model.

*Queries Requiring Access to Backend Database (SQL):* The Karma provenance service is primarily intended as a provenance recording and querying system, and has limited support for recording generic metadata and annotations. Annotations present in provenance activities are stored in the relational backend and optionally returned as part of the provenance data model. However, the Karma service API does not support queries over annotations. Instead, they are queried upon by directly executing SQL statements over the database. Queries Q4, Q5, Q6, and Q8 are answered in this manner. Figure 2 shows the UML diagram of the relational database schema and the queries are defined over this.

Query Q4 is to find all invocations of the AlignWarp service that had input parameter '-m 12' as annotation and executed on a Monday. The SQL query given below retrieves the invocation IDs for the matching invocations that can then be used to get the process provenance through the service API. The invocation ID is a combination of the entity IDs of the client (invoker) and and the service (invokee) that are part of the invocation. The annotations are part of the Notification (Activity) table and we search for the input parameter within the ServiceInvoked activity. The backend database is MySQL; so the day the process ran is specified using MySQL date-time functions. In this case, Monday corresponds to the 2nd day of the week.

```
SELECT
 invokee.workflow_id, invokee.service_id, invokee.workflow_node_id,
 invokee.workflow_timestep, invoker.workflow_id, invoker.service_id,
 invoker.workflow_node_id, invoker.workflow_timestep
FROM
 invocation_state_table invocation, entity_table invokee,
 entity_table invoker, notification_table notifications
WHERE
 invokee.entity_id = invocation.invokee_id AND
 invoker.entity_id = invocation.invoker_id AND
 notifications.source_id = invocation.invokee_id AND
 notifications.notification_type = 'ServiceInvoked' AND
 invokee.service_id =
   'urn:qname:http://.../karma/challenge06:AlignWarpService' AND
 notifications.notification_xml LIKE
   '%<ModelMenuNumber>12</ModelMenuNumber>%' AND
 DAYOFWEEK(invocation.request_receive_time) = 2;
```

Once invocation IDs are returned by the query for the matching invocations, the `getProcessProvenance` provenance service API method is called with these invocation IDs to return the process provenance for each of the invocations.

*Queries Out of Scope (Unsupported):* Query Q9 is one that is purely based on annotations and the Karma service API does not support such complex queries on annotations. While it is possible to perform incremental SQL queries combined with service API calls to answer this query, such queries are not scalable and deemed outside the scope of the Karma provenance system. Karma is part of the LEAD Cyberinfrastructure project and, as in other such projects, there are information services such as the myLEAD personal catalog and Resource Catalog [24] that record generic metadata about data products and services. We expect such systems to answer the bulk of the annotation queries and the provenance is just one piece in the information integration landscape of the virtual organization.

## 4. DISCUSSION AND CONCLUSION

The provenance systems participating in the challenge are categorized according to a taxonomy [17]. Using this matrix, the Karma framework falls under the event driven approach (C2.2), with the provenance activities forming events. In addition, it also uses the data derivation information present in the activities to establish the causal relationship of provenance. This hybrid approach is used by several other systems that participated in the challenge, such as VisTrails [19], RWS [18], and COMAD [3], while some like JP [15] rely on just an event mechanism.

Karma does not require the workflow to be provided (C2.1) and all provenance information is collected at workflow runtime. This allows Karma to handle *adaptive workflows* whose definition change as they running [12]. Some systems like REDUX [1], SDG [20], myGrid [27], and Wings [14] require the abstract workflow be provided and provenance is collected by attaching runtime information to the abstract workflow template.

We also record *client and service views* of provenance similar to OPA [16]. Many systems use a relational databases or XML for representing provenance. We leverage the advantages of both (C1.3). However, we do not support semantic markup using RDF or OWL that Mindswap [13] and NCSD2K/NCSI [10] use. While direct SQL queries are used to answer some challenge queries, this is not the expected mode of usage and the Karma service API is preferred (C1.4).

An abstraction of *nested workflows* similar to VDL [4] is used for grouping provenance metadata in Karma, allowing control of the depth of the provenance to retrieved (C2.7). We also use the concept of *immediate* and *deep provenance* to configure the provenance depth by time, similar to ZOOM [5]. While data of flexible granularity is allowed by Karma using global IDs (C2.6), it is collected at the level of the service. Systems like PASS [21] and ES3 [8] allow for a lower level of provenance collection at the Unix process and through file handles.

Though annotation queries are not in the scope of Karma, we were able to address all but one of the challenge queries (C2.3). Additional queries that can form part of future challenges include those that make use of the nested workflow abstraction, queries over workflows containing loops, queries over dynamic, adaptive workflows, and queries comparing client and service views of workflows.

We see opportunities to optimize the Karma query interface, especially to tackle some deep provenance queries that currently require client post-processing. These can be easily moved into the service and exposed through the service API, substituting multiple web service calls by the client with more efficient SQL queries by the service. This goes for most of the queries that require client post-processing; the main trade-off is loosing the present simplicity of the service API. While annotations are important to get the complete context to provenance, as shown by the several interesting annotation

queries in the challenge, having them as part of the provenance service blurs the line between generic metadata catalogs and provenance services. This behooves the need for meta information services that can integrate provenance and external metadata such as annotations and provide uniform query capability over both.

**REFERENCES**


1. Barga RS, Digiampietri LA. Automatic Capture and Efficient Storage of eScience Experiment Provenance. *Concurrency and Computation:Practice and Experience,* in this issue, 2007.
2. Bose R, Frew J. Lineage retrieval for scientific data processing: a survey. *ACM Computing Surveys* 2005; **37**(1)1–28.
3. Bowers S, McPhillips T, Ludaescher B. Provenance Model for Collection-Oriented Scientific Workflows. *Concurrency and Computation:Practice and Experience,* in this issue, 2007.
4. Clifford B, Foster I, Hategan M, Stef-Praun T, Wilde M, Zhao Y. Tracking provenance in a virtual data grid. *Concurrency and Computation:Practice and Experience,* in this issue, 2007.
5. Cohen-Boulakia S, Cohen S, Davidson S. Addressing the provenance challenge using ZOOM. *Concurrency and Computation:Practice and Experience,* in this issue, 2007.
6. Czajkowski K, Kesselman C, Fitzgerald S, Foster I. Grid information services for distributed resource sharing. In *HPDC,* Redondo Beach, CA, 2001.
7. Droegemeier KK, et al. Service-oriented environments for dynamically interacting with mesoscale weather. *Computing in Science and Engineering* 2005; **7**(6)12–29.
8. Frew J, Metzger D, Slaughter P. Automatic Capture and Reconstruction of Computational Provenance. *Concurrency and Computation:Practice and Experience,* in this issue, 2007.
9. First Provenance Challenge, Karma results page. http://twiki.ipaw.info/bin/view/Challenge/Karma [28 November 2006].
10. Futrelle J, Myers J. Tracking Provenance Semantics in Heterogeneous Execution Systems. *Concurrency and Computation:Practice and Experience,* in this issue, 2007.
11. Gannon D, Plale B, et al. Service oriented architectures for science gateways on grid systems. In *ICSOC,* Amsterdam, Netherlands, 2005.
12. Gannon D, Plale P, Marru S, Kandaswamy G, Simmhan YL, Shirasuna S. Dynamic, adaptive workflows for mesoscale meteorology. In *Workflows for eScience: Scientific Workflows for Grids,* Gannon D, et al. (eds.). Springer, 2006.
13. Golbeck J, Hendler J. A Semantic Web Approach to Tracking Provenance in Scientific Workflows. *Concurrency and Computation:Practice and Experience,* in this issue, 2007.
14. Kim J, Deelman E, Gil Y, Mehta G, Ratnakar V. Provenance trails in the Wings/Pegasus system. *Concurrency and Computation:Practice and Experience,* in this issue, 2007.
15. Krenek A, Sitera J, Matyska L, Dvorak F, Mulac M, Ruda M, Salvet Z. gLite Job Provenance – a Job-Centric View. *Concurrency and Computation:Practice and Experience,* in this issue, 2007.
16. Miles S, Groth G, Munroe S, Jiang S, Assandri T, Moreau L. Extracting causal graphs from an open provenance data model. *Concurrency and Computation:Practice and Experience,* in this issue, 2007.
17. Moreau L, Ludascher B, et al. The first provenance challenge. *Concurrency and Computation:Practice and Experience,* in this issue, 2007.
18. Podhorszki N, Ludaescher B, Altintas I, Bowers S, McPhillips T. Recording Data Provenance for Kepler Scientific Workflows. *Concurrency and Computation:Practice and Experience,* in this issue, 2007.
19. Scheidegger C, Koop K, Santos E, Vo H, Callahan S, Freire J, Silva C. Tackling the Provenance Challenge One Layer at a Time. *Concurrency and Computation:Practice and Experience,* in this issue, 2007.
20. Schuchardt K, Gibson T, Stephan E, Chin Jr. G. Applying Content Management to Automated Provenance Capture. *Concurrency and Computation:Practice and Experience,* in this issue, 2007.
21. Seltzer M, Holland DA, Braun U, Muniswamy-Reddy K. PASS-ing the Provenance Challenge. *Concurrency and Computation:Practice and Experience,* in this issue, 2007.
22. Simmhan YL, Plale P, Gannon G. A Survey of Data Provenance in e-Science. *SIGMOD Record* 2005; **34**(3)31–36.
23. Simmhan YL, Plale P, Gannon G. A Framework for Collecting Provenance in Data-Centric Scientific Workflows. In *ICWS,* Chicago, IL, 2006.
24. Simmhan YL, Pallickara SL, Vijayakumar NN, Plale P. Data management in dynamic environment-driven computational science. In *IFIP WoCo9,* Prescott, AZ, 2006.
25. Singh G, et al. A metadata catalog service for data intensive applications. In *Supercomputing,* Phoenix, AZ, 2003.
26. Slominski A. Adapting BPEL to Scientific Workflows. In *Workflows for e-science,* Taylor I, et al. (eds.). Springer, 2006.
27. Zhao J, Goble C, Stevens R, Turi D. Mining Taverna's Semantic Web of Provenance. *Concurrency and Computation:Practice and Experience,* in this issue, 2007.