Project 1706 netKarma User Guide

Sep 30, 2010

This user guide describes the NetKarma Provenance Collection tool and helps developers interested in using the persistent Axis2 Web Service and XML API to interact with the NetKarma repository.

1. NetKarma Provenance Collection Tool

NetKarma is a tool and repository for capturing and accessing the workflow of GENI experiments at multiple layers in the PlanetLab stack, including slice creation, topology of the slice, operational status, and links to measurement data. NetKarma consists of Karma provenance collection tool and persistent web service as an underlying infrastructure, and the Advanced Message Queuing Protocol standards compliant RabbitMQ messaging service used to ingest notifications to Karma database. The persistent Axis2 web service is available at *http://bitternut.cs.indiana.edu:42816/axis2/-services/KarmaService*. It has a WSDL access API so provenance can be stored and retrieved programmatically. NetKarma Provenance Repository resides on a server in the GENI Meta-Operations Center (GMOC), located at Indiana University. Further information about the NetKarma software is available at *http://pti.iu.edu/d2i/provenance_netkarma.*

2. Persistent Web Service for NetKarma

NetKarma utilizes an Axis2 based persistent Web Service, Karma Service, to enable interaction with the provenance repository. The Karma Web Service provides both publish and query XML API. We discuss the semantics of a XML API for the Karma Service particularly focusing on the query capabilities. We introduce the details of the XML API in the following section followed by a discussion on usage scenarios to give an idea for how one query the repository using the XML API.

2.1.The Karma Service Query Interface

Use the Query XML API to pose queries to the Karma service for provenance metadata. The API consists of both macro-level and object-level (or entity-level) queries. A macro-level query is a complex query, which walks the database, tracing provenance relationships back through time in order to construct a graph. It returns the annotated graph in XML format. An object-level query locates information about specific entities matching the conditions specified in passing arguments. See the Query XML API in Table 1. The table gives the types and target data too.

Karma Inquiry XML API	**Target Data Type in Karma DB**	**Query Type**
findAbstractService(…)	Higher-level registry provenance data	Object-level query
getAbstractServiceDetail(…)		
findAbstractMethod(…)		
getAbstractMethodDetail(…)		
findAbstractDataProduct(…)		
getAbstractDataProductDetail(…)		
findService(…)	Lower-level execution layer provenance data	
getServiceDetail(…)		
findClient(…)		
getClientDetail(…)		
findDataProduct(…)		
getDataProductDetail(…)		
findMethodInvocation(…)		

getMethodInvocationDetail(…)		
getAnnotationDetail(…)	Higher- and lower-level provenance data	
getWorkflowGraph(…)	Higher- and lower-level provenance data	Macro-level query
getProvenanceHistory(…)	Higher- and lower-level provenance data	Macro-level query
getNodeCountOfGraph(…)	Higher- and lower-level provenance data	Macro-level query
compareGraphs(…)	Higher- and lower-level provenance data	Macro-level query

Table 1. API for Query Interface

Software that allows people to explore and examine large quantities of data requires browse capabilities. The browse pattern characteristically involves starting with some broad information, performing a search, finding general result sets and then selecting more specific information for drill-down. The Karma Service Query XML API supports the browse pattern by way of the API calls involving "find" operations ("find_xx" APIs). These calls form the search capabilities provided by the API and are matched with summary return structures that return overview information about the registered provenance information associated with the query API and the search criteria specified in the inquiry. Each instance of the core Karma information model data structures has a uniqueURI which is one of the items in the summary information retrieved by find_xx APIs. Given such a ID, it is easy to retrieve the full detailed provenace metadata for the corresponding instance by passing the ID to the relevant get_xx API.

2.2.The Karma Service Query XML API

Find abstractService: The find abstractService API call locates specific services or workflows matching the conditions specified in the arguments.

```
findAbstractService ()
        Params:   -> method name
                  -> workflow ID []
                  -> sub service name
                  -> next service name
                  -> annotation []
                  -> InputDataProductID
                  -> OutputDataProductID
                  -> serviceName (regexp)
                  -> InstanceOfService []
        Return:    -> uniqueID[] {DB internal primary key}
                  -> service name[]
```

Get abstractServiceDetail: The get abstactServiceDetail API call returns the abstactService structure corresponding to each of the uniqueID values specified in the arguments. It is used to retrieve service/workflow metadata associated with a unique identifier.

```
getAbstractServiceDetail ()
        Params:  -> uniqueID[]
        Return:   -> AbstractService[]
```

Find abstractMethod: The find abstractMethod API call returns a list of abstractMethod structure matching the conditions specified in the arguments. It is used to find the abstractMethod entity elements. An abstractMethod entity defines metadata about a function or procedure that executes a computational or data transformation task.

```
findAbstractMethod ()
        Params:   -> service ID []
                  -> hasInputDataProductID
                  -> hasOutputDataProductID
                  -> annotation []
                  -> parameter name
                  -> methodName[] (regexp)
                  -> hasInstanceOfMethod

        Return:   -> uniqueID[]
                  -> method name[]
```

Get abstractMethodDetail: The getAbstractMethodDetail API call returns the abstractMethod structure corresponding to each of the uniqueID values specified in the arguments.

```
getAbstractMethodDetail ()
        Params:  -> uniqueID[]
        Return:  -> AbstractMethod[]
```

Find abstractDataProduct: The findAbstractDataProduct API call returns a list of abstractDataProduct structure matching the conditions specified in the arguments.

```
findAbstractDataProduct ()
        Params:   -> InputOfServiceID
                  -> OutputOfServiceID
                  -> isInputOfMethodID
                  -> isOutputOfMethodID
                  -> annotation []
                  -> dataProductName[] (startsWith, EndsWith, ... regexp)
                  -> hasInstanceOfDataProduct
        Return:   -> uniqueID[]
                  -> data product name[]
```

Get abstractDataProductDetail: The getAbstractDataProductDetail API call returns the abstractDataProduct structure corresponding to each of the uniqueID values specified in the arguments.

```
getAbstractDataProductDetail ()
        Params:   -> uniqueID[]
        Return:   -> AbstractDataProduct[]
```

Find service: The findService API call locates specific instances of services or workflows matching the conditions specified in the arguments. This API call locates provenance data captured in execution level.

```
findService ()
        Params:   -> name
                  -> host
                  -> architecture
                  -> initialization time
                  -> termination time
                  -> isSuccess
                  -> annotation[]
                  -> subServiceName
                  -> prevServiceName
        Return:   -> uri[]
                  -> service name[]
```

Get serviceDetail: The getServiceDetail API call returns the service structure corresponding to each of the uniqueID values specified in the arguments.

```
getServiceDetail ()
        Params:  -> uniqueURI []
        Return:  -> service detail
```

Find client: The findClient API call locates specific instances of clients matching the conditions specified in the arguments. The client is an entity which initiates workflows or services. It could be a user or workflow engine. This API call locates provenance data captured in execution level.

```
findClient ()
        Params:  -> name
                 -> service ID []
                 -> workflow name
                 -> annotation[]
        Return:  -> uri[]
                 -> client name[]
```

Get clientDetail: The getClientDetail API call returns the client structure corresponding to each of the uniqueID values specified in the arguments.

```
getClientDetail ()
        Params:  -> uniqueURI[]
        Return:  -> client detail
```

Find dataProduct: The findDataProduct API call returns a list of dataProduct structure matching the conditions specified in the arguments. It is used to find the dataProduct entity elements.

```
findDataProduct ()
        Params:  -> name
                 -> value
                 -> instanceOfDataProduct
                 -> annotation[]
        Return:  -> uniqueID[]
                 -> data product name[]
```

Get dataProductDetail: The getDataProductDetail API call returns the dataProduct structure corresponding to each of the uniqueID values specified in the arguments.

```
getDataProductDetail()
        Params:  -> uniqueURI[]
        Return:  -> data product detail[]
```

Find methodInvocation: The findMethodInvocation API call returns a list of method structure matching the conditions specified in the arguments. It is used to find the methodInvocation entity elements.

```
findMethodInvocation()
        Params:  -> parameters from findAbstractMethod()
                 -> timestamp
                 -> request status
                 -> request receiveTime
                 -> request sendTime
                 -> response status
                 -> response receiveTime
```

```
                    -> response sendTime
                    -> annotation[]
          Return:   -> method invocation ID []
                    -> method invocation name []
```

Get methodInvocationDetail: The getMethodInvocationDetail API call returns the methodInvocation structure corresponding to each of the uniqueID values specified in the arguments.

```
getMethodInvocationDetail ()
          Params:   -> uniqueID[]
          Return:   -> methodInvocation[]
```

Get annotationDetail: The getAnnotationDetail API call returns the annotation structure corresponding to each of the uniqueID values specified in the arguments.

```
getAnnotationDetail ()
          Params:   -> uniqueID []
          Return:   -> Annotation []
```

Get worflowGraph: The getWorkflowGraph API call returns a graph (either in XML or RDF format) corresponding to the workflow uniqueID value specified in the arguments.

```
getWorkflowGraph ()
          Params:   -> workflow-uniqueID
                    -> information_detail_level
                    -> return_format (rdf or xml)
          Return:   -> graph (either in rdf or xml)
```

Get provenanceHistory: The getProvenanceHistory API call returns the provenance trace (either in XML or RDF format) corresponding to the data object uniqueID value specified in the arguments.

```
getProvenanceHistory ()
          Params:   -> data-object-uniqueID
                    -> information_detail_level
                    -> time_range (long value)
                    -> return_format (rdf or xml)
          Return:   -> graph (either in rdf or xml)
```

Get nodeCountOfGraph: The getNodeCountOfGraph API call returns the number of nodes in a given workflow corresponding to the workflow URI value specified in the arguments.

```
getNodeCountOfGraph ()
          Params:   -> workflow_URI
          Return:   -> # of nodes in given workflow graph
```

Compare Graphs: The compareGraphs API call returns a list of differences in two different workflow graphs as (name, value) pairs. As arguments, it takes the workflow URIs of the two workflows that needed to be compared.

```
compareGraphs ()
          Params:   -> workflow_URI, workflow_URI
          Return:   -> list of differences in (name, value) pairs
```

3.Example Usage Scenario with Karma Service XML API

To facilitate testing of the netKarma tool, we apply provenance capture to an experiment running on GENI network and capture provenance information. We focused on provenance collection from Gush (http://gush.cs.williams.edu/trac/gush) to capture provenance information available at the experiment level about a particular experiment and its execution. Gush is an extensible execution management system for GENI. To facilitate testing of the prototype, we used Twister (http://www.iterativemapreduce.org), a parallel, iterative version of the MapReduce to execute a crawling application using breath-first search through a large-scale random graph. By utilizing MapReduce programming framework, the application explores the nodes of the same level of the graph in parallel, and then goes to the nodes in next levels iteratively. This way, it is able to process breadth-first graph search in parallel. We utilized PlanetLab, where we run Gush to deploy and execute the Twister experiment. Gush requires that users describe their experiments or computation in an XML document. It uses this document to locate and access the remote resources in PlanetLab. In its execution flow, Gush contacts a host to deploy a twister server, which then reads a configuration file and internally connects to other hosts where the application needs to run. In this experimental study, provenance of both successful and failed executions (because of improper setup of the application) captured.

To find lineage information for a given data product (artifact) in a Twister experiment run, we use Karma Service XML API. For example, let say, this query start with a call getProvenanceHistory, passing a data-product-uniqueID. This returns a graph (either in XML or RDF) of a workflow (or collection of worklflows) used to generate the data-product under investigation. In turn, resulting graph may be used as input by a visualization tool to trace the lineage of an artifact. The getProvenanceHistory method allows user to set a time-range to constrain how far back in time the request should go to attempt to build a provenance trace. It also allows distinction in the level of provenance information to be returned.