Komadu Provenance Collection Framework

Version 1.0

User Guide

April 8th, 2014

This user guide contains instructions for building and installing the Komadu service, v1.0, which provides core capability to store provenance events and provide lineage graphs. Komadu Provenance Tool is licensed under Apache License, Version 2.0 (http://www.apache.org/licenses/LICENSE-2.0). The code is copyrighted and copyright owned by The Trustees of Indiana University. Komadu is a product of the Data to Insight Center at Indiana University. See http://pti.iu.edu/d2i/provenance for more information.

Table of Contents

- Table of Contents
- Overview
- Key Features
- Software Dependencies
- Setup Komadu Database
- Configuration Properties
  - Required Properties
  - Optional Properties
- Building Komadu Source
- Set-up Tomcat and deploy Komadu as a Web Service
- Executing Test Cases through Web Service Client
- Host Komadu as Standalone Server using RabbitMQ
  - Install RabbitMQ
  - Build and Run Komadu Service as Standalone Server
- Komadu Messaging Client
  - Build Komadu Messaging Client
  - Execute Messaging Client
- Komadu Ingest/Query API
  - Ingest API
  - Inferencing Relationships
  - Query API
- Visualization
  - Software Dependencies
  - How to Use

Overview

According to W3C PROV specification, the term “provenance” is defined as follows.

“Provenance is information about entities, activities, and people involved in producing a piece of data or thing, which can be used to form assessments about its quality, reliability or trustworthiness.”

When this is applied in the context of digital scientific data, provenance become a critical component in broadening, sharing and reusing of scientific data. Provenance captures the information needed to attribute ownership and determine the quality of a particular data set.

Komadu is a W3C PROV (http://www.w3.org/TR/prov-dm/) compliant standalone provenance collection tool that can be added to an existing cyberinfrastructure for the purpose of collecting and visualizing provenance data. It is the successor of Karma provenance tool (http://d2i.indiana.edu/provenance_karma) which is based on OPM. Komadu provides a Web Services based API for both provenance collection and querying collected data. Provenance collection is driven by notifications which represents a particular event related to some activity, entity or agent. Query API can be mainly used to find details about a particular activity, entity or agent and to generate the provenance graph for a particular activity, entity or agent.

Visualization of provenance data is more useful with support for manipulating very large structures, for displaying different views and for interactivity. This can help a user to navigate their experiment information with a mental map of what is going on in the experiment, to compare different experimental runs quantitatively, and to do model selection with an effective collaboration between the user and the discovery system. Komadu comes with a command line tool which converts a generated graph into a CSV (Comma Separated Values) file and that can be imported into most of the visualization tools Cytoscape.

Key Features

1. **Support for W3C PROV specification:** Komadu is fully compliant with the W3C PROV specification and generates provenance graphs according to the PROV-XML schema (http://www.w3.org/TR/2013/NOTE-prov-xml-20130430/).
2. **Easy to use API:** Komadu ingest and query APIs are nicely aligned with W3C PROV standards. Therefore it's very easy to understand and use for someone who is familiar with the specification.
3. **Ability to collect provenance information of any application:** Unlike Karma, Komadu is not tightly coupled to workflows and scientific provenance collection. Komadu API uses generic terms and operations such that any type of provenance can be captured and queried.
4. **Contextless graph generation:** Karma generates provenance graphs based on a context workflow URI which must be shared across all workflows/services within the system. This limitation is no longer there in Komadu and it can generate graphs starting from a given node by following their relationships.
5. **Support for Activity, Entity and Agent graphs:** Komadu can generate the provenance graph starting from any node which can be an Activity, Entity or Agent. This allows the user to track lineage of any component that he/she is interested in.
6. **Backward compatibility for context graphs:** Komadu provides the support for context based graphs for workflows to make it backward compatible with Karma.
7. **Asynchronous notification processing:** Komadu comes with an asynchronous notification processing mechanism which makes the tool faster and more responsive. All incoming raw notifications are stored in the database first and a background thread pool processes those notifications later. Behaviour of the thread pool can be controlled through a set of parameters in komadu.properties file.
8. **Customizable cache expiration:** All generated graphs are cached in the database to be reused within the cache interval. This cache interval can be configured through a parameter in komadu.properties file.
9. **Visualization of graphs:** Komadu comes with a tool which converts the generated XML based PROV graph into a CSV file which can be imported to most of the visualization tools like Cytoscape.
10. **Test cases to cover all ingest and query API operations:** Komadu comes with a set of tests which cover almost all the supported operations. User can follow these to develop their own Komadu client.

Software Dependencies

Komadu v1.0 depends on following software packages. Therefore these must be installed before setting up Komadu. Please follow the given links to find specific installation instructions for each software package.

1. Apache Maven 3.0 or higher
   http://maven.apache.org/

2. MySQL database server 5.1
   http://dev.mysql.com

3. MySQL Connector/JDBC 5.1 or higher
   http://dev.mysql.com/downloads/connector/j/

4. JDK 1.6 or higher
   http://www.oracle.com/technetwork/java/javase/downloads/index.html

5. Apache Tomcat 6.0.x or higher
   http://tomcat.apache.org/

6. Prov ToolBox 0.4.0
   https://github.com/lucmoreau/ProvToolbox/releases

7. Apache Axis2 1.6.2
   http://axis.apache.org/axis2/java/core/download.cgi

8. RabbitMQ Server 3.3.1 or higher (Only if you use messaging)
   http://www.rabbitmq.com/

9. Apache ANT v1.6 or higher (for building Komadu messaging client)
   http://ant.apache.org/

Setup Komadu Database

Before hosting the service, we have to set up the Komadu database in MySQL. Please follow these instructions in the given order.

1. Log into mySQL as an administrator:

```
mysql -u root -p
```

2. Create a database, preferably named "komadu"

```
CREATE DATABASE komadu;
```

3. Create login credentials for komadu and grant permissions

```
GRANT ALL ON komadu.* TO 'komaduuser'@'localhost' IDENTIFIED BY 'komadupwd';
GRANT SELECT ON mysql.proc TO 'komaduuser'@'localhost';
```

In the example above, 'komaduuser' and 'komadupwd' are just examples. DO NOT use these verbatim; choose something more secure instead, especially for the password.

This example also assumes that the database is hosted on the same node as Komadu service. If this is not the case, please specify the host onto which Komadu is hosted:

```
GRANT ALL ON komadu.* TO 'komaduuser'@'dedicated-node.iu.edu' IDENTIFIED BY
'komadupwd';
GRANT SELECT ON mysql.proc TO 'komaduuser'@'dedicated-node.iu.edu';
```

4. Use the provided database schema definition to define tables. The schema definition file can be found in the “service-core/config/” directory of the Komadu service distribution, and is named komadu_db_schema.sql.

```
cd ${service-core}
mysql -u root -p komadu < config/komadu_db_schema.sql
```

5. If the database is hosted on a different node than Komadu, following EXECUTE grant should be given to the stored procedure.

```
GRANT EXECUTE ON PROCEDURE komadu.PR_OBJECT_LOCK
TO 'komaduuser'@'dedicated-node.iu.edu';
```

Configuration Properties

The Komadu server depends on a number of properties for the correct operation as well as performance tweaking. The distribution package contains a sample properties file, which can be found in service-core/config/komadu.properties. Please use this sample file to configure Komadu according to the deployment environment. Below is the detailed explanation of each of these properties.

Required Properties

These properties must be properly set regardless of how Komadu is hosted.

log4j.properties.path - path to a log4j.properties file. Komadu uses log4j to log information. A sample log4j.properties file is also provided with the distribution package, in the service-core/config/ directory. This log4j.properties file defines two separate loggers, komadulog and komaduconsole. komadulog is a rolling file appender that writes log messages to a series of files on disk, whereas komaduconsole is a console appender that writes log messages to screen. Please modify log4j.properties to change settings such as the name and location of the log file etc.

database.location – URI of komadu database

database.username – username for logging into komadu database

database.password – password for logging into komadu database

cache.expiration – maximum time for cached graphs to be valid (specified in minutes)

conn.pool.init.size – initial size of komadu database connection pool. Creating database connection is expensive. Komadu uses a connection pool to deliver faster performance.

conn.pool.max.size – maximum size of komadu database connection pool.

conn.time.to.live.ms – number of milliseconds a database connection in the pool lives before being replaced by a new connection. This is to ensure the database connections in the pool do not timeout. Please choose a moderate number. If the number is too small, each connection gets replaced too frequently and causes performance overhead; on the other hand, if the number is too large, the connection may timeout and also cause performance penalty.

async.processor.thread.count - number of asynchronous raw (unprocessed) notification processors to run. Komadu asynchronously processes ingested notifications to avoid server overload.

raw.notif.process.batch.size – maximum number of raw (unprocessed) notifications to fetch from the database for each access.

raw.notif.cache.size - number of raw notifications each asynchronous processor takes after they are fetched from the database.

The following formula may serve as a guideline for specifying the above 3 properties:
raw.notif.cache.size = raw.notif.batch.size / async.processor.thread.count

attribute.definition.scope.count - number of known annotation definition scope settings. This property must be followed by *N* pairs of **attribute.definition.property.***x* and **attribute.definition.scope.***x*

attribute.definition.property.*x* - name or URI of annotation property *x*, where 1<=*x*<=*N, N* being the value given to **attribute.definition.scope.count**

attribute.definition.scope.*x* - scope of annotation property *x*, where 1<=*x*<=*N*. The value of this property **must be one of the following tokens**:

PROV_ANNOTATION - this annotation is defined by PROV
KOMADU_ANNOTATION - this annotation is defined by Komadu
EXTERNAL_SOURCE - this annotation is defined by some other external source

Optional Properties

These properties are applicable only if Komadu uses event service bus (RabbitMQ) communication.

The following 5 properties are used to connect to a RabbitMQ Server. If the instance of RabbitMQ already exists, contact the system admin for details.
messaging.username - RabbitMQ Username
messaging.password - RabbitMQ Password
messaging.hostname - Hostname that hosts RabbitMQ Server
messaging.hostport - Port number on the RabbitMQ Server
messaging.virtualhost - The virtual host is created for administrative purposes. Each connection (and all channels inside) must be associated with a single virtual host. Each virtual host comprises of its own namespace, a set of exchanges, message queues and all associated objects. The default value is "/".

The following 3 properties are used to configure how to send the Notifications to Komadu Server.

messaging.exchangename - A message routing agent. It can be durable (our system uses durable), temporary, and auto-deleted. Each message is delivered to each qualifying queue. The default value here is “KomaduExchange”.

messaging.queuename - Named “Weak FIFO” buffer. The default value is “KomaduQueue”.

messaging.routingkey -- In our implementation, we use direct exchange type. Same outingkey is used on both publisher and subscriber sides. The default value here is “KomaduKey” The following 2 properties are used for the Karma Server to reconnect to the RabbitMQ Server in case of failures.

messaging.retry.interval -- Komadu server will wait for N seconds (retry interval) before each retry if connection between itself and RabbitMQ Server is lost. The default value for N is 5.

messaging.retry.threshold -- Komadu Server will retry up to X times (retry threshold) to create the connection between itself and RabbitMQ Server. The default value for X is 5.

Building Komadu Source

Komadu provides a set of maven scripts to build the project. Before building, you have to install maven as mentioned above.

1. Download Prov-ToolBox 0.4.0 from the link below.
   https://github.com/lucmoreau/ProvToolbox/releases

   Then build only “prov-model” and “prov-xml” modules by moving into those directories, using the following command. Only those two modules are used by Komadu.

   ```
   mvn clean install
   ```

2. Download Komadu source distribution.

3. Edit the services.xml file found under service-core-aar/src/main/resources/META-INF and set the correct path for the "komadu.properties.file.path" parameter. Default “komadu.properties” file can be found under service-core/config/ directory. You may modify it with proper values for all required properties and provide the full path in the services.xml.

4. Build Komadu using the following command. We have to skip tests when building for the first time because we haven’t deployed the service yet.

   ```
   mvn clean install -Dmaven.test.skip=true
   ```

Set-up Tomcat and deploy Komadu as a Web Service

You have installed Tomcat and built Komadu source in the previous steps. Now we can deploy Komadu as an Axis2 service on Tomcat.

1. Download Apache Axis2 1.6.2 war file.

2. Copy it to {your_tomcat_home}/webapps directory and start Tomcat once and then shut it down. This will extract the axis2.war and create a "axis2" directory under "{your_tomcat_home}/webapps/" directory.

3. Copy {komadu_checkout_path}/service-core-aar/target/komadu-service-1.0.aar file to {your_tomcat_home}/webapps/axis2/WEB-INF/services directory.

4. Download and copy mysql-connection-java-5.1.x-bin.jar to {your_tomcat_home}/lib.

5. Copy following jar files into {your_tomcat_home}/lib.
   {your_home}/.m2/repository/org/openprovenance/prov/prov-model/0.4.0/prov-model-0.4.0.jar
   {your_home}/.m2/repository/org/openprovenance/prov/prov-xml/0.4.0/prov-xml-0.4.0.jar
   {your_home}/.m2/repository/org/jvnet/jaxb2_commons/runtime/0.4.1.5/runtime-0.4.1.5.jar
   {your_home}/.m2/repository/commons-lang/commons-lang/2.6/commons-lang-2.6.jar

6. Create a directory {your_tomcat_home}/endorsed and copy following 2 jar files into it. You can find these jars in your local maven repository.
   {your_home}/.m2/repository/javax/xml/bind/jaxb-api/2.2.4/jaxb-api-2.2.4.jar
   {your_home}/.m2/repository/com/sun/xml/bind/jaxb-impl/2.1.10/jaxb-impl-2.1.10.jar

7. Start Tomcat. If you can see the following log on the console without any errors, you have successfully deployed the Komadu Axis2 service on Tomcat.

   [INFO] Deploying Web service: komadu-service-1.0.aar - file:/home/isuru/software/apache-tomcat-7.0.29/webapps/axis2/WEB-INF/services/komadu-service-1.0.aar

Executing Test Cases through Web Service Client

Komadu comes with a set of test cases which ingests notifications covering almost all the relationships supported by Komadu. And then it executes queries to generate following types of lineage graphs.

- Context Workflow Graph
- Non-Context Activity Graph
- Non-Context Entity Graph
- Non-Context Agent Graph

Follow the steps below to execute the tests through maven.

1. Start Tomcat which already contains the deployed Komadu service.

2. Open the following file.
   {komadu_checkout_path}/axis2-client-core/src/test/resources/komadu.client.properties

3. In that file, set the proper value for “komadu.service.url” property. If you are running Tomcat on localhost using it’s default port 8080, the URL will be as follows.
   http://localhost:8080/axis2/services/KomaduService

4. Now go to {komadu_checkout_path}/axis2-client-core directory and run the tests using the following command.

   ```
   mvn clean install -o
   ```

   You’ll see the XML graphs generated for each query operation printed during the maven build. On Tomcat logs, you’ll be able to see [DEBUG] logs for all operations performed.

5. You can have a look at the test code in the following file to get an idea on how to write a client for Komadu to ingest notifications and execute query operations.

   {komadu_checkout_path}/axis2-client-core/src/test/java/edu/indiana/d2i/komadu/axis2/client/KomaduTests.java

Host Komadu as Standalone Server using RabbitMQ

This section is on how to host Komadu as a standalone server using RabbitMQ for messaging.

Install RabbitMQ

Install RabbitMQ by following the installation guide at the URL given below. There are different instructions for different platforms.

http://www.rabbitmq.com/download.html

You can use the RabbitMQ management plugin to easily manage your RabbitMQ installation and check statistics.

https://www.rabbitmq.com/management.html

Build and Run Komadu Service as Standalone Server

If you haven't already build Komadu using the top level maven script, build it using the following command.

```
mvn clean install -Dmaven.test.skip=true
```

Then move into the "service-core" directory and execute following command.

```
./build-sandalone.sh
```

This will create a "bin" directory and you'll be able to find the generated KomaduServer.sh script inside it. Before executing the server, you have to properly configure the "service-core/config/komadu.properties" file by following the instructions given in "Configuration Properties" section. You have the specify all messaging related properties there. After that is done, execute the following commands to run the server.

```
cd {komadu_checkout_path}/service-core/bin
./KomaduServer.sh ../config/komadu.properties
```

This will start the Komadu messaging server. Default log file (komadu.log) can be found in the bin directory after starting the server.

Komadu Messaging Client

This section is on how to use Komadu Messaging Client when the Komadu Server is hosted as standalone server and using RabbitMQ for messaging. This section assumes that you have already set up the Komadu Messaging server.

Build Komadu Messaging Client

Execute the following command to run the ant script which generates shell scripts for sending notifications and queries to RabbitMQ server.

```
cd {komadu_checkout_path}/client-core-messaging
ant
```

Execute Messaging Client

Before executing the Messaging Client, you have to properly configure the "client-core-messaging/config/komadu.properties" file with all messaging properties. Refer to "Configuration Properties" section above for more information on each property.

The following 2 properties are used to validate the format of incoming notifications and query requests against Komadu ingest and query schema. You have to configure those as well in komadu.properties file.

komadu.ingest.schema - Komadu ingest schema xsd file path
komadu.query.schema - Komadu query schema xsd file path

After that, you can start sending notifications to Messaging Server. We provide two ways of doing it; one through shell scripts and the other one through ant.

Notification Ingest

```
./bin/sendNotification.sh config/komadu.properties ${path_to_notification_xml_file}
```

or

```
ant sendNotification –Dnotification=${path_to_notification_xml_file}
```

Query

```
./bin/query.sh config/komadu.properties ${path_to_notification_xml_file} ${path_to_output}
```

or

```
ant query –Dquery=${path_to_query_xml_file}
```

To run any main method in the komadu client package

```
./bin/KomaduRun.sh <Main Class> <Arguments>
```

or

```
ant run –DrunClass=<Main Class> -DrunArgs=”<Arguments>”
```

In addition to these, if you are trying to invoke the messaging client API from your own code, you can use the API found in “client-core-messaging/build/lib/komadu-client.jar”.

Komadu Ingest/Query API

Komadu comes with a very easy to use API. Currently it's exposed through a Web Service interface. You can use the komadu-axis2-client-core-1.0.jar as a dependency to create your own Komadu client and ingest notifications into Komadu.

Ingest API

Ingest API is made very easy to understand by following the W3C PROV standard in designing the API. Here we explain each API operation in detail. First six operations are used to ingest a relationship between two components.
Please have a look at the complete schema in komadu_ingest_schema.xsd found under axis2-client-core/config directory to find all schema elements.

1. addAgentActivityRelationship

   This operation is used to ingest a relationship between an Agent and an Activity. Schema element that represents the input message for this operation is shown below. Here we provide information about the Agent and Activity and then provide details of the relationship. Only relationship which is possible between an Agent and an Activity is 'Association'.

```
<complexType name="agentActivityType">
      <sequence>
         <element name="agent" type="komadu:agentType"/>
         <element name="activity" type="komadu:activityType"/>
         <element name="association" type="komadu:associationType"/>
      </sequence>
</complexType>
```

2. addAgentEntityRelationship

   This operation is used to ingest a relationship between an Agent and an Entity. Schema element that represents the input message for this operation is shown below. Here we provide information about the Agent and Entity and then provide details of the relationship. Only relationship which is possible between an Agent and an Entity is 'Attribution'.

```
<complexType name="agentEntityType">
      <sequence>
         <element name="agent" type="komadu:agentType"/>
         <element name="entity" type="komadu:entityType"/>
         <element name="attribution" type="komadu:attributionType"/>
      </sequence>
</complexType>
```

3. addActivityEntityRelationship

   This operation is used to ingest a relationship between an Activity and an Entity. Schema element that represents the input message for this operation is shown below. Here we provide information about the Activity and Entity and then provide details of the relationship. Then a relationship must be chosen out of 5 possible relationships between an Activity and an Entity.

```
<complexType name="activityEntityType">
      <sequence>
         <element name="activity" type="komadu:activityType"/>
         <element name="entity" type="komadu:entityType"/>
         <choice>
            <element name="usage" type="komadu:usageType"/>
            <element name="generation" type="komadu:generationType"/>
            <element name="start" type="komadu:startType"/>
            <element name="end" type="komadu:endType"/>
            <element name="invalidation" type="komadu:invalidationType"/>
         </choice>
      </sequence>
</complexType>
```

4. addAgentAgentRelationship

   This operation is used to ingest a relationship between 2 Agents. Schema element that represents the input message for this operation is shown below. Here we provide information about the 2 Agents and then provide details of the relationship. Only relationship which is possible between 2 Agents is ‘Delegation’.

```
<complexType name="agentAgentType">
      <sequence>
         <element name="delegateAgent" type="komadu:agentType"/>
         <element name="responsibleAgent" type="komadu:agentType"/>
         <element name="delegation" type="komadu:delegationType"/>
      </sequence>
</complexType>
```

5. addActivityActivityRelationship

   This operation is used to ingest a relationship between 2 Activities. Schema element that represents the input message for this operation is shown below. Here we provide information about the 2 Activities and then provide details of the relationship. Only relationship which is possible between 2 Activities is 'Communication'.

```
<complexType name="activityActivityType">
      <sequence>
         <element name="activity1" type="komadu:activityType"/>
         <element name="activity2" type="komadu:activityType"/>
         <element name="communication" type="komadu:communicationType"/>
      </sequence>
</complexType>
```

6. addEntityEntityRelationship

   This operation is used to ingest a relationship between 2 Entities. Schema element that represents the input message for this operation is shown below. Here we provide information about the 2 Entities and then provide details of the relationship. Then a relationship must be chosen out of 5 possible relationships between 2 Entities.

```
<complexType name="entityEntityType">
      <sequence>
         <element name="entity1" type="komadu:entityType"/>
         <element name="entity2" type="komadu:entityType"/>
         <choice>
            <element name="derivation" type="komadu:derivationType"/>
            <element name="revision" type="komadu:revisionType"/>
            <element name="quotation" type="komadu:quotationType"/>
            <element name="primarySource" type="komadu:primarySourceType"/>
            <element name="alternate" type="komadu:alternateType"/>
```

```
            <element name="specialization" type="komadu:specializationType"/>
            <element name="membership" type="komadu:membershipType"/>
         </choice>
      </sequence>
</complexType>
```

7. addAttributes

   This operation can be used to add Attributes to any existing Activity, Entity or Agent in the system. This is useful when we want to add extra Attributes which were not added when the object was initially added. Schema element that represents the input message for this operation is shown below.

```
<complexType name="addAttributesType">
      <sequence>
         <element name="objectType" type="komadu:objectEnumType"/>
         <element name="objectID" type="anyURI"/>
         <element name="entityType" type="komadu:entityEnumType"
minOccurs="0"/>
         <element name="attributes" type="komadu:attributesType"/>
         <element name="addAttributeTimestamp" type="dateTime" minOccurs="0"/>
         <element name="notificationTimestamp" type="dateTime"/>
      </sequence>
</complexType>
```

   Here, the objectType can be an Activity, Entity or Agent. Then there must be an objectID which is a unique URI to identify the object. If the object is an Entity, entityType must be specified and it can take values File, Block, Collection and Generic. Then the set of new attributes has to be provided.

Inferencing Relationships

Komadu is able to build certain relationships by performing inferences through the set of relationships provided by the user. Following are 2 examples of such relationships.

1. Communication

   If the user has sent a ‘generation’ relationship between Activity a1 and Entity e1, and a ‘usage’ relationship between Activity a2 and Entity e1, then Komadu automatically builds a ‘communication’ relationship between Activity a1 and Activity a2 according to the W3C PROV specification. If the user has already sent that relationship too, this inference won’t be done and users one will be used.

2. Derivation

   If the user has sent a ‘generation’ relationship between Activity a1 and Entity e1, and a ‘used’ relationship between Activity a1 and Entity e2, then Komadu automatically builds a ‘derivation’ relationship between Entity e1 and Entity e2 according to the W3C PROV specification. If the user has already sent that relationship too, this inference won’t be done and users one will be used.

Query API

Komadu Query API mainly contains two kinds of operations. A set of operations to generate different kinds of graphs and another set of operations to find objects already ingested into Komadu.
Please have a look at the complete query schema in komadu_query_schema.xsd found under axis2-client-core/config directory to find all schema elements.

1. getActivityGraph

   This operation returns a provenance graph constructed by starting from the activity which is identified by the given activity URI. No context information needed. Komadu constructs the graph expanding each node by considering relationships.

2. getEntityGraph

   This operation returns a provenance graph constructed by starting from the entity which is identified by the given entity URI. No context information needed for this too.

3. getAgentGraph

   This operation returns a provenance graph constructed by starting from the agent which is identified by the given agent URI. No context information needed for this too.

4. getContextWorkflowGraph

   This operation is supported in Komadu to make it compatible with Karma. User has to provide a context workflow URI for this query. Komadu will return the provenance graph which only contains the activities and related entities and agents within the given context.

5. findService

   Can be used to find a service or set of services which matches with a given set of attributes.

6. getServiceDetail

   Returns all the details about a set of services which is identified by a given set of URIs or an internal ids.

7. findEntity

   Can be used to find an entity or set of entities which matches with a given set of attributes.

8. getEntityDetail

   Returns all the details about a set of entities which is identified by a given set of URIs or an internal ids.

Visualization

Komadu comes with a tool which transforms a PROV xml document into two CSV files: an edge list CSV and a node list CSV. These CSV files can be imported and visualized by many standard visualization tools including Cytoscape and Gephi.

Transformation tool can be found under ‘visualization’ directory in Komadu package. PROV graph XMLs generated by Komadu can be used as the input for this tool. There are some sample PROV graphs under visualization/samples directory.

Software Dependencies

Before you start to use this tool, you have to install a XSLT engine. This is because, essentially, this tool comprises a list of XSLT documents. These XSLT documents should be executable from any standard XSLT engine. For example, we've installed SAXON-HE:

http://saxon.sourceforge.net/

How to Use

Open a terminal and go to the 'visualization' directory and execute the following command.

```
java net.sf.saxon.Transform -s:<your-PROV-file> -xsl:xslt/pipeline_xml2csv.xsl
```

We've provided some example PROV files in 'samples' directory. For example, you can transform those using the following command.

```
java net.sf.saxon.Transform -s:sample1.xml -xsl:xslt/pipeline_xml2csv.xsl
```

After it finishes, you'll see two CSV files: 'edges.csv' and 'nodes.csv'. For information about how to import these two CSV files into visualization tools such as Cytoscape, please read their manuals, for example:

http://wiki.cytoscape.org/Cytoscape_3/UserManual#Cytoscape_3.2BAC8-UserManual.2BAC8-Creating_Networks.Import_Networks_from_Unformatted_Table_Files

Below is a brief summary of importing CSV files into Cytoscape:

1. Import Network From File
   Choose 'edges.csv'. Check 'Show TextFile Import Options', make sure you select 'Comma' as delimiter and check the option 'Transfer first line as column name'. Select

the right 'Source' (2nd column), 'Interaction' (1st), and 'Target' (3rd). Left clicks to enable the columns that you want to import as attributes.

2. Import Table From File
   Choose 'nodes.csv'. Check 'Show Text File Import Options', make sure you select 'Comma' as delimiter and check the option 'Transfer first line as column name'. Check 'Show Mapping Options' and the 'primary key' should be default to 'id' already.

3. Apply Visual Style
   Import vismap through 'File-->Import-->Vismap File', and select 'style/prov-visMap.xml'. Apply the 'PROV' visual style by right click on the network from the network control panel and choose 'Apply Visual Style', then select 'PROV' from the list. We recommend to choose 'Layout-->Hierarchical Layout' as the first layout algorithm.

A sample PROV graph is given below.