Provenance Manager: PROV-man
an Implementation of the PROV Standard

Ammar Benabadelkader

BioLab group meeting

AMC - KEBB, 14 January 2014
Outlines

- Motivation
- State-of-the-art
- PROV-man
  - The Approach, the data model, the API
- Usage examples
Motivation for Provenance

• Discoveries in modern science involve:
  – large amount of data, many people, varied tools, etc.

• Good scientific practice dictates that findings should be:
  – Traceable and reproducible

• Data provenance tracking approaches can play a major role in addressing many of these challenges.

Data provenance proposes ways to capture, manage, and use of provenance information.
State-of-the-Art

• Origin
  – from the French *provenir*, "to come from",
  – originally mostly used for works of art,
  – currently, used in a wide range of fields, including archaeology, paleontology, archives, manuscripts, printed books, and e-science

In this presentation, we refer to Provenance in e-science as *data provenance*
State-of-the-art

• Early efforts
  – < 1990: unstructured logs and temp files,
  – since 90’s: information systems (e.g. DICOM, LIMS, ELN)
  – 2000+: data provenance become more prominent
  – 2007: Open Provenance Model (OPM)
  – 2013: PROV
Today: PROV

- Few implementations
  - Application specific
- PROV-man
  - Open/generic framework
PROV-man: The Approach

• **PROV-man design requirements:**
  – permanent storage of provenance data and approaches for optimization,
  – Easy access to provenance data
  – Support utilities for data sharing
  – Easy deployment of the framework in various use cases

• **PROV-man framework consist of:**
  – Database implementing PROV-DM
  – Application Programming Interface (API)
PROV-DM: The Data Model

- **PROV Data Model consists of:**
  
  - Core data types (*Entity*, *Activity*, and *Agent*);
  
  - A set of *Relations* between the core data types;
  
  - A set of *Attributes* that could be defined for each of the core data types and Relations,

  - A *Document* grouping all the above.
PROV-DM: The Data Model
PROV-man: The Data Model

- **PROV-man data model of:**
  - Optimized data model
  - Relational DBMS (MySQL)

- XML-configuration file to specify:
  - the underlying database,
  - connection parameters, and
  - tuning parameters
PROV-man: The API

*PROV-man* API provides an interface for the management of provenance data that preserves the semantics and richness defined by PROV and makes the PROV-man data model transparent to the user.
PROV-man: The API

• PROV-man open architecture provides:
  – **classes with methods** to manipulate provenance data;
  – **interfaces** implementing utility functions;
  – **back-end database** that serves as a main repository for storing provenance data;
  – **Object-relational mapping**.
• **PROV-man Classes:**
  - Implements the *SET* and *GET* methods for PROV concepts;
  - *PROVmanFactory* provides a set of additional methods to create provenance data, using a human readable notation;
PROV-man: The API

• **PROV-man Classes:**
  – Implements the *SET* and *GET* methods for PROV concepts;
  – *PROVmanFactory* provides a set of additional methods to create provenance data, using a human readable notation;

• **PROV-man Interfaces:**
  – implements utility functions for data sharing and interoperation:
    • `toDB(document)`
    • `toXML(document)`
    • `toProvN(document)`,
    • `toOWL2(document)`,
    • `toGraphviz(document)`,
    • `toGraph(document, format).`
Example 1: on-line newspaper article

- presented in PROV-PRIMER:
  - publishes an article with a chart about crime statistics
  - based on data, with values composed by geographical regions.
  - The data uses different namespace prefixes to identify the its source (e.g. exb, exn, exc, and exg)
public static Document createSample() {
    ProvmanFactory provFactory = new ProvmanFactory();
    Collection<Entity> entities = new ArrayList<Entity>();
    Collection<Agent> agents = new ArrayList<Agent>();
    Collection<Activity> activities = new ArrayList<Activity>();
    Collection<Relation> relations = new ArrayList<Relation>();
    Document document = new Document();
    // Entities
    Entity e2 = new Entity(); e2.setId("exc:dataSet1");
    Entity e3 = new Entity(); e3.setId("exc:regionList1");
    Entity e4 = new Entity(); e4.setId("exc:composition1");
    Entity e5 = new Entity(); e5.setId("exc:chart1");
    entities.add(e2); entities.add(e3);
    entities.add(e4); entities.add(e5);
    // Activities
    Activity act1 = new Activity(); act1.setId("exc:compose1");
    ActivityAttributes act1Attr = new ActivityAttributes();
    act1Attr.setKey("Status"); act1Attr.setValue("Done");
    act1.getAttribute().add(act1Attr);
    activities.add(act1);
    Activity act2 = new Activity(); act2.setId("exc:illustrate1");
    ActivityAttributes act2Attr = new ActivityAttributes();
    act2Attr.setKey("Status"); act2Attr.setValue("Ongoing");
    act2.getAttribute().add(act2Attr);
    activities.add(act2);
    // Agents
    Agent agent1 = new Agent(); agent1.setId("exc:derek");
    provFactory.addAgentAttributes(agent1, "prov:type", "Person");
    provFactory.addAgentAttributes(agent1, "foaf:givenName", "Derek");
    provFactory.addAgentAttributes(agent1, "foaf:mbox", "derek@example.org");
    agents.add(agent1);
Agent agent2 = new Agent(); agent2.setId("exc:chartgen");
provFactory.addAgentAttributes(agent2, "prov:type", "Organization");
AgentAttributes agAttr = new AgentAttributes();
agAttr.setKey("foaf:Name"); agAttr.setValue("Chart Gen. Inc");
agent2.getAttributes().add(agAttr);
agents.add(agent2);

// relationships
WasAssociatedWith waw1 = new WasAssociatedWith();
waw1.setId("waw1"); waw1.setActivity(act2); waw1.setAgent(agent1); waw1.setPlan(e2);
relations.add(waw1);
ActedOnBehalfOf abo1 = provFactory.newActedOnBehalfOf("abo1", agent1, agent2);
relations.add(abo1);
WasAttributedTo wat1 = provFactory.newWasAttributedTo("wat1", e5, agent1);
relations.add(wat1);
Used used1 = provFactory.newUsed("used1", act1, e2, "prov:role", "exc:dataToCompose");
relations.add(used1);
Used used2 = provFactory.newUsed("used2", act1, e3, "prov:role", "exc:regionsToAggregateBy");
relations.add(used2);
WasGeneratedBy wgb1 = provFactory.newWasGeneratedBy("wgb1", e4, act1, "prov:role", "exc:composedData");
relations.add(wgb1);
Used used3 = provFactory.newUsed("used3", act2, e4);
relations.add(used3);
WasGeneratedBy wgb2 = provFactory.newWasGeneratedBy("wgb2", e5, act2);
relations.add(wgb2);
WasDerivedFrom wdf1 = provFactory.newWasDerivedFrom("wdf2", e5, e4, "prov:type", "prov:Revision");
relations.add(wdf1);

document = provFactory.newProvGraph("provenance of an online newspaper article", activities, entities, agents, relations);

return document;
public static void main(String args[]) {
    ProvmanPersistence PROVman = ProvmanPersistence.instance();
    PROVman.init();
    Document doc = PROVman.createSample();
    PROVman.toDB(doc);
    PROVman.toGraph(doc, "png");
}
Using PROV-man

- To use the PROV-man framework, please take the following steps:
  - Download PROV-man
  - Create DB
  - Deploy and test the API library
  - Define mapping Application -> PROV concepts
  - Integrate into Application
PROV-man: Download

- Download the PROV-man API form sourceforge:
  - [http://sourceforge.net/projects/provman/?source=directory](http://sourceforge.net/projects/provman/?source=directory)

- Releases
- Support
- Documentation
PROV-man: Create DB

• Create the PROV-man database using the PROV-man DDL script:
  – src/resources/PROV-man.ddl

• Update the database configuration file
  – src/resources/hibernate.cfg.xml
PROV-man: Deploy and Test

• Create a simple Java program that executes

```java
public static void main(String args[]) {
    ProvmanPersistence PROVman = ProvmanPersistence.instance();
    PROVman.init();
    Document doc = PROVman.createSample();
    PROVman.toDB(doc);
    PROVman.toGraph(doc, "png");
}
```
**PROV-man: Define mapping**

- Define the mapping between application and PROV concepts
- Decide on what data to collect and to which depth

<table>
<thead>
<tr>
<th><strong>PROV concept</strong></th>
<th><strong>workflow counterpart</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Document</td>
<td>Workflow execution (experiment)</td>
</tr>
<tr>
<td>Entity</td>
<td>- Input data</td>
</tr>
<tr>
<td></td>
<td>- output result</td>
</tr>
<tr>
<td>Agent</td>
<td>- Users of the platform</td>
</tr>
<tr>
<td></td>
<td>- Organization</td>
</tr>
<tr>
<td></td>
<td>- Data analysis tool</td>
</tr>
<tr>
<td>Activity</td>
<td>jobs executed</td>
</tr>
<tr>
<td>Relation</td>
<td>Input/output relationship</td>
</tr>
<tr>
<td>Attributes (all)</td>
<td>Metadata for all above</td>
</tr>
<tr>
<td>- Document</td>
<td>- Experiment attributes</td>
</tr>
<tr>
<td>- Entity</td>
<td>- Input/output data</td>
</tr>
<tr>
<td>- Agent</td>
<td>- attributes</td>
</tr>
<tr>
<td>- Activity</td>
<td>- Agent attributes</td>
</tr>
<tr>
<td>- Relation</td>
<td>- Jobs attributes</td>
</tr>
<tr>
<td></td>
<td>- Relationships attributes</td>
</tr>
</tbody>
</table>
Alternatives:

- **Integrate** into the implementation of the Java application.
- **Data collector** as an external software module using logs, database, etc.
- **Extract** the provenance data directly from the application’s database
- Combine the above
Example: Provenance for the Neuroscience Gateway

• Major part of the data retrieved for the Neuroscience database (catalogue):
  – information about executed workflows, their jobs, status, input data and output results.

• Detailed information about each executed job parsed from the log files on the grid:
  – start time, end time, computing node, operating system on the computing node, etc.)

• Collector automatically triggered
Implementation

<table>
<thead>
<tr>
<th>PROV concept</th>
<th>workflow counterpart</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document</td>
<td>Workflow execution (experiment)</td>
</tr>
</tbody>
</table>
| Entity       | - Input data  
              - Output result |
| Agent        | - Users of the platform  
              - Organization  
              - Data analysis tool |
| Activity     | jobs executed |
| Relation     | Input/output relationship |
| Attributes (all) | Metadata for all above  
                      - Experiment attributes  
                      - Input/output data attributes  
                      - Agent attributes  
                      - Jobs attributes  
                      - Relationships attributes |
**PROV-man: your gain?**

- **Developers:**
  - Reduce development time;
  - Enhance and facilitate the utilization of the standardization;
  - Customize format according to user preferences and requirements.

- **Users (scientists):**
  - Provenance data for their experiments (out of the box)
    - Validated, repeated, shared, trusted, proven experiments
  - Interoperability
Thanks!

Links

- **PROV-man**: [http://www.bioinformaticslaboratory.nl/twiki/bin/view/EBioScience/PROVMan](http://www.bioinformaticslaboratory.nl/twiki/bin/view/EBioScience/PROVMan)
- **PROV-Primer**: [http://www.w3.org/TR/2013/NOTE-prov-primer-20130430/](http://www.w3.org/TR/2013/NOTE-prov-primer-20130430/)
- **PROV-DM**: [http://www.w3.org/TR/2013/REC-prov-dm-20130430/](http://www.w3.org/TR/2013/REC-prov-dm-20130430/)
- **PROV-O**: [http://www.w3.org/TR/2013/REC-prov-o-20130430/](http://www.w3.org/TR/2013/REC-prov-o-20130430/)
- **PROV-N**: [http://www.w3.org/TR/2013/NOTE-prov-sem-20130430/](http://www.w3.org/TR/2013/NOTE-prov-sem-20130430/)
- **Resource Description Framework (RDF)**: [http://www.w3.org/TR/rdf-mt/](http://www.w3.org/TR/rdf-mt/)
- **Graphviz** - Graph Visualization Software: [www.graphviz.org](http://www.graphviz.org)
Discussion / Questions