

# UNICORE Summit 2015

## Provenance Tracking in UNICORE André Giesler



# Content



- Definition
- Motivation
- Preliminary considerations
- Design decisions
- Current status

# Provenance

## What we understand by that...



- Tracing the origins of data
- Recording information during processing
  - But it's not logging
  - Tracking relations between data!
- Enabling reproducibility
  - To re-run processes
  - To ensure the conformity of data and processes

**The provenance of an information is the  
history of its production**

# Motivation

## Typical issues from the scientific environment



- In what kind of way did the PhD student, who left us years ago, obtain the results in the XYZ simulation (compilers, software versions, scripts, parameters, libs, used supercomputer)?
  - How was the data generated that is used as an input file in our compute job?
  - Which values got Variable A during its lifetime in that workflow?
  - Did the script used in a Unicore workflow use the right algorithm?
  - On which supercomputer and environment did that job run?
  - Provide me with all workflows having that specific user annotation.
  - Compare the parameters of two similar jobs
- 
- Allow any conceivable backtracking by Provenance



# Motivation Partners

- We are collaborating with some Neuromedicine Institutes at Forschungszentrum Jülich in
  - creating complex workflows for image processing in human brain research
  - generating workflows in the field of electrophysiological data analysis
  - transferring data and recording its life cycle
- All collaborators have a strong focus on workflows and its data provenance but don't have software solutions combining both features
- UNICORE needs some provenance functionality to fulfill these requirements

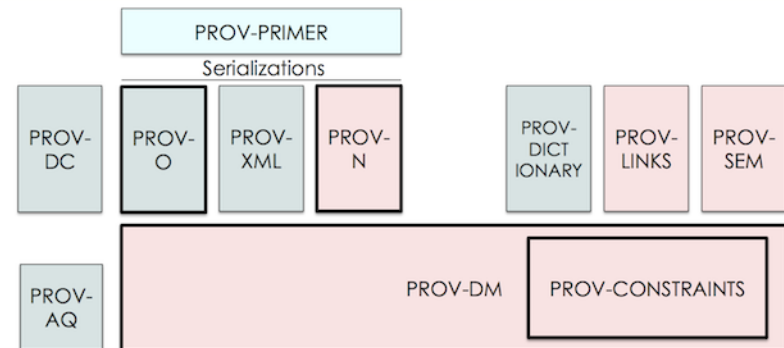


- Provenance description should be written, ideally, in a generally accepted standardized format/ontology
- It must be ensured that job and workflow structure, files (references), literals, logical structures as loops, metadata, user annotations can be mapped without effort to the chosen provenance format
- A suitable storage backend and mode is required to store the provenance data and to allow efficient queries

# Preliminary considerations

## Provenance Description Format

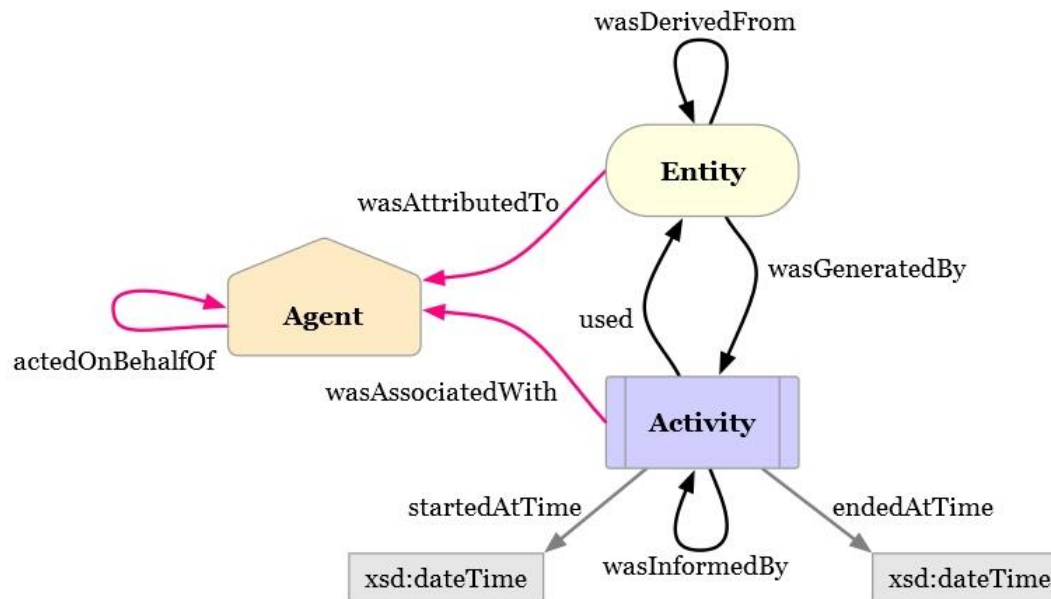
- There were some efforts in recent years to standardize provenance
- The Open Provenance Model (OPM) initiated 2006 provides an abstract model with ontologies for web, biology, workflows (adopted by Kepler, Taverna)
  - Enables exchange of provenance information
  - Allows developing and build tools
  - digital representation in RDF triples (subject predicate object)
- Successor is the PROV family published by W3C in 2013
  - Provides a basic vocabulary and data model
  - Available in different notations (RDF, XML, JSON)




# Preliminary considerations

## PROV

- Subjects and objects are Agents, Entities, and Activities
- Representation as a directed graph, where predicates are the edges



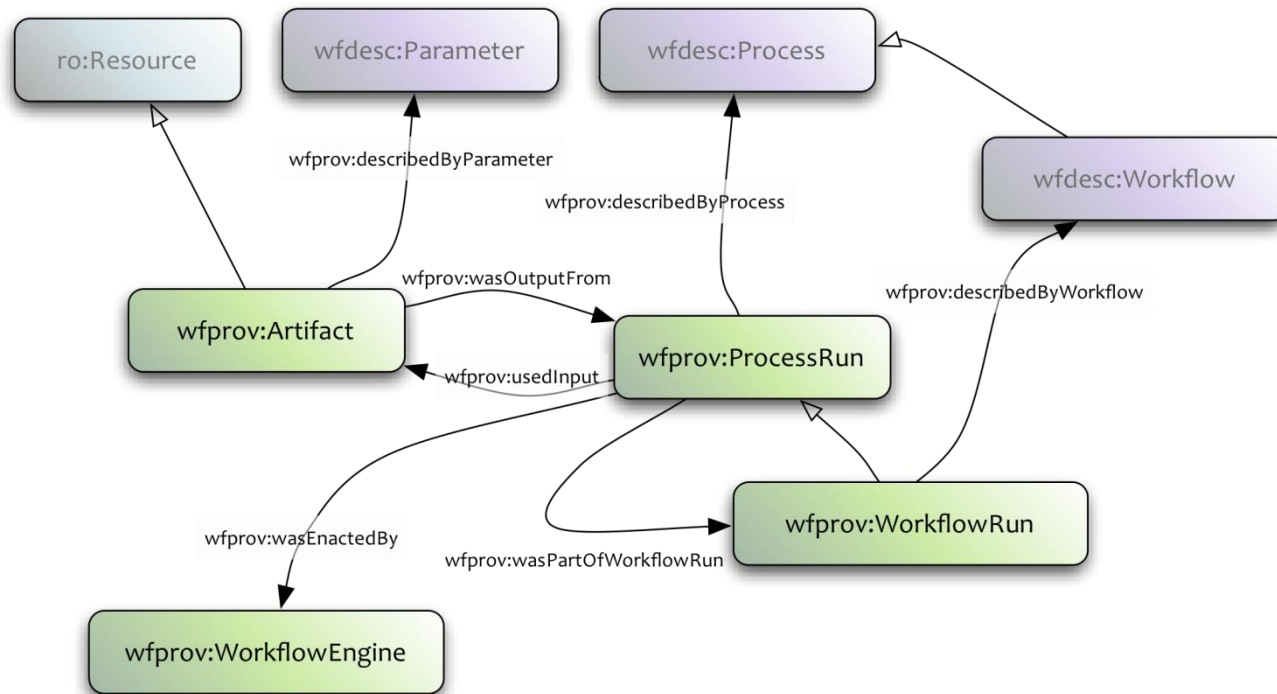
- For example, Activities generate Entities  
RDF notation  :Data-Entity prov:wasGeneratedBy :Job-Activity



# Preliminary considerations

## Wf4ever-Project

- Provides ontologies to describe a workflow centric Research Object
- Extends basic PROV ontology
- Static Workflow description (wfdesc:;) and dynamic Workflow execution (wfprov:;) provenance



# Preliminary considerations

## Storage Backend and Querying



- Provenance is different from other forms of meta data
- It is based on the relationship among objects and their logical sequence
- In practice Provenance forms graph
- Unicore Workflow model is a graph...
- As a consequence:
  - The data model used for provenance should provide a natural representation for directed graphs
  - Any query language should have direct, simple, and straightforward support for reasoning about graphs and paths through them.

# Preliminary considerations

## Which Storage backend for Graphs



- Robust production-grade relational databases are widespread
  - However, the relational model is the complete antithesis of a graph-oriented model
  - Representing graphs in an RDBMS requires tables of nodes and edges, and creating paths by joining these lists to itself repeatedly
- XML might be a suitable hierarchical back-end representation for graphs, but XPath/XQuery are not appropriate for querying provenance
- RDF databases/ triple-stores (SPARQL query language) are in general a good option for graphs since an RDF triple is a graph.
- In comparison, graph databases have a more generalized structure than triple-stores
  - Optimized for graph traversals (e.g. shortest path queries).
  - With RDF triple stores, the cost of traversing an edge tends to be logarithmic.

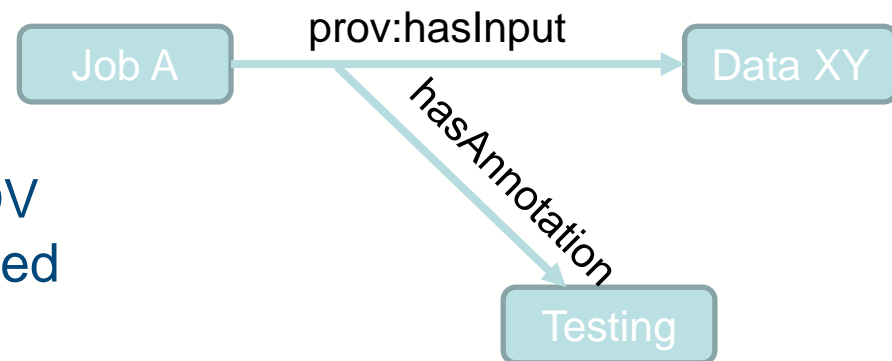
# Design Decisions

## Storage Backend

- We haven determined the graph database Neo4j as the storage backend for provenance data
- Neo4j is widely used and currently the most popular Graph-DBMS (No.21 in world wide database ranking <http://db-engines.com/de/ranking>)
- Query language Cypher is a declarative, SQL-inspired language for describing patterns in graphs
- Provides a browser-based visual representation of the graph data
  - Filter mechanism
  - Construct easily Cypher queries on the data
  - No need to implement another UI for querying



- PROV and the Wf4ever-PROV extension seem to be a very good choice for mapping Jobs, Processes, Workflows, data, and the relations among them to a machine and human readable format
- Benefits
  - Well-defined provenance description
  - Interoperable format for exchanging purposes
- Todo
  - PROV is notated in RDF triples (Subject, predicate, object)
  - While Neo4j is notated as a Property Graph



- Conclusion: A mapping from PROV to property graph notation is needed

- Evaluation phase started in May 2015
- Creating knowledge about Provenance and carrying out a market analysis
- Implementation/testing started a few weeks ago
  - Defining which information should be tracked to Provenance (done)
  - Defining a PROV pattern structure (ongoing, PROV extension needed)
  - Adding a provenance layer to Unicore Server modules (just started)
  - Setting up Neo4j database and model the PROV pattern structure as a graph (just started)
  - Involving INM partners in the implementation process to ensure the acceptance of the product
- Team: Myriam Czekala (implementation), André Giesler, Björn Hagemeyer (advisory function)

Thank you for your attention!