H2020-EINFRA-2017

EINFRA-21-2017 - Platform-driven e-infrastructure innovation

DARE [777413] “Delivering Agile Research Excellence on European e-Infrastructures”

Data Lineage Services I

Project Reference No	777413 — DARE — H2020-EINFRA-2017 / EINFRA-21-2017
Deliverable	D-3.3 Data Lineage Services I M18
Work package	WP3: Large-scale Lineage and Process Management
Tasks involved	T3.2, T3.3
Type	DEM: Demonstrator, pilot, prototype
Dissemination Level	PU = Public
Due Date	30/06/2019 Deadline extended to 31/07/2019 in agreement with PO
Submission Date	31/7/2019
Status	Final Draft
Editor(s)	Alessandro Spinuso (KNMI)
Contributor(s)	
Reviewer(s)	Christian Pagé (CERFACS)
Document description	Overview of the first release of the Lineage Services.

Document Revision History

Version	Date	Modifications Introduced Modification Reason	Modified by
0.1	6/11/2018	Outline and inclusion of Lineage API	Alessandro Spinuso
0.3	19/05/2018	Reorganisation and Extended Sections	Alessandro Spinuso
0.8	28/06/2019	Section on S-PROV Model, Conclusions and overall refinements	Alessandro Spinuso
0.9	15/07/2019	Internal Review	Christian Pagé
1.0	23/07/2019	Addressed reviewer's comments in Sections 2,3	Alessandro Spinuso

Executive Summary

This document describes the current release of the DARE lineage service API, which is deployed in DARE test-bed and used by the application workflows developed by WP6 and WP7. The activities on the API conducted by WP3 focused on improving its documentation and deployment mode, besides developing new methods and updating its software dependencies. Section 2 introduces the underlying provenance model, derived by generic and common standards, while Section 3 describes the use cases and how they map to the methods of the API. We have reported in another deliverable, D3.7 a collection of tools already exploiting the web-service.

Table of Contents

List of Terms and Abbreviations

Abbreviation	Definition
PROV	W3C Standard for Provenance Representation
S-PROV	PROV extension for lineage representation of of streaming operators.

1 Introduction

Provenance and data-lineage information offer insights that interest different roles, from the domain-scientists to the community manager. In data-intensive computations this information can be overwhelming, hiding latent but significant evidence of a method's effectiveness and efficiency towards meeting required goals. The API offers ways of interactively accessing and visualising the provenance according to general purpose use cases, offering detailed interactive navigation of single executions, as well as tunable perspectives involving data, people and infrastructures across multiple workflow runs.
The API introduced in this report allowed the realisation of data-lineage exploration tools. These show how computations can be monitored and evaluated interactively at different levels of detail, visually combining computation and scientific metadata with users' processes. The database technology and the adopted representation proved sufficiently flexible to accommodate the rapid implementation of the use cases. The underlying provenance model S-PROV, for data-intensive and stateful operations, which we briefly describe in this deliverable, accommodates complex lineage patterns and represents details about the mapping of the abstract workflow to its distributed and concurrent execution.

2 S-PROV: Resource Mapping and Stateful Operators

S-PROV, thanks to the explicit representation of stateful operations and of the distribution of the computation, accommodates complex lineage patterns and represents details about the mapping of the abstract workflow to its distributed and concurrent execution. The model, shown in Figure 1, is built by importing and further specialising concepts introduced by the PROV[1] data-model in combination with ProvONE[2]. PROV is intended as a conceptual framework offering machine understandable descriptions of records that describe with contextual metadata people, institutions, entities, and activities involved in producing, influencing, or delivering a piece of information. In PROV, an *Activity* informs another *Activity* by means of an exchange of information, which consists of the *Entities* that are used and generated when accomplishing a specific task. Respectively they are associated with and attributed to an *Agent*, who can perform or delegate the task to other agents. ProvONE, offers instead an extension point to accommodate the provenance representation of more precisely characterised workflow computational processes.

[1] http://www.w3.org/TR/prov-dm/

[2] https://purl.dataone.org/provone-v1-dev

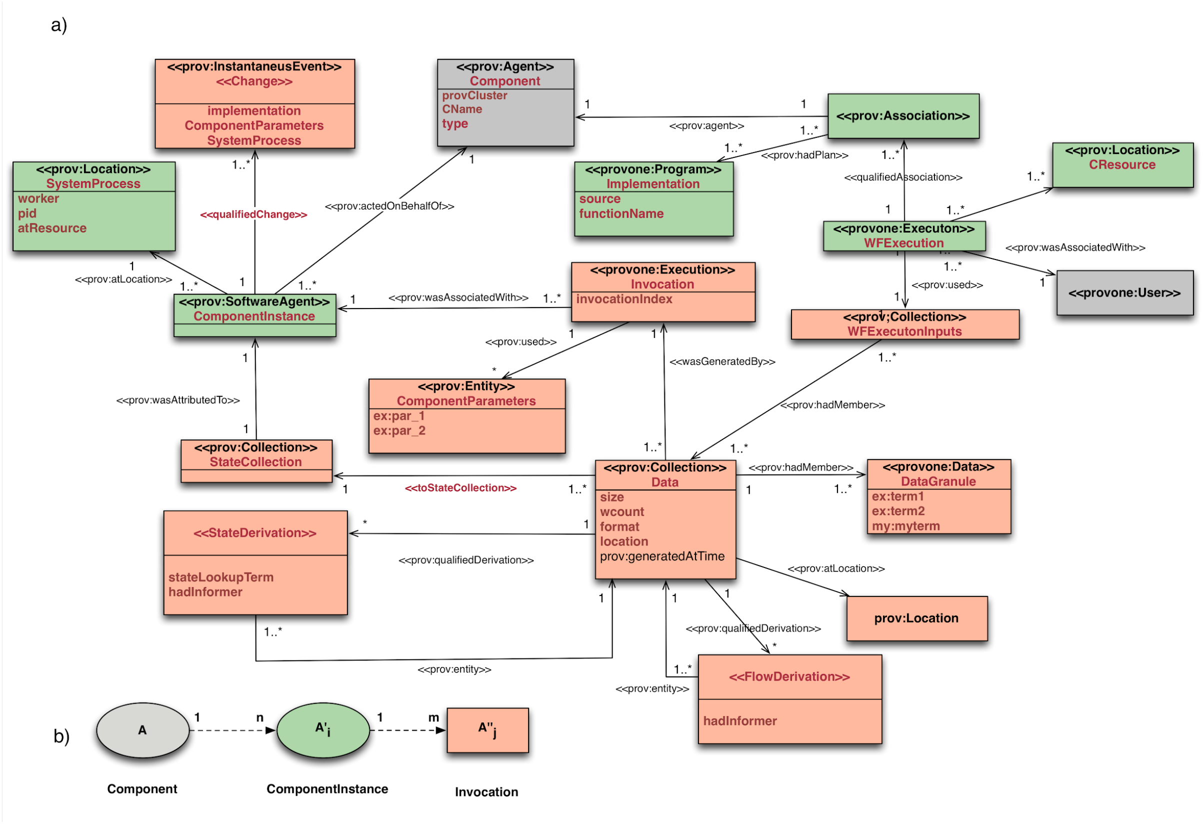

Figure 1: (a) S-PROV provenance model. Colour coding indicates the following: (grey) elements for abstract and prospective provenance; (green) concrete workflow elements and state; (red) execution elements and fine grain dependencies. Extensions of PROV and ProvONE are indicated for each class. (b) After the abstract workflow is deployed to the underlying resources, each Component is mapped into several *ComponentInstances* that perform many Invocations to execute the workflow task on the incoming data. Components and *ComponentInstances* are *prov:Agents*.

More specifically, while ProvONE represents the structure of a *provone:Workflow* as a graph of interconnected entities of type *provone:Program*. These are executed according to the computational model specified by a *provone:Controller*. In S-PROV we extend the description of the abstract workflow by introducing a new class, *Component* (the abstract workflow step), which extends the basic PROV class *prov:Agent*. In S-PROV, a *Component* delegates the execution of a program to multiple instances of the program itself. We represent the instances in S-PROV by introducing a new class *ComponentInstance* that extends the PROV class *prov:SoftwareAgent*. Making this semantics explicit enables us to represent in the provenance traces detailed information about the execution of parallel operators. Information includes updates to their internal state with intermediate and reusable data. ProvONE, through the concept of *provone:Workflow*, which is also a program, and the relationship *provone:subprograms*, offers support for workflow encapsulation, which is an important and reusable feature of the model. Here the activity class of *provone:Execution* applies to workflows, as well as their internal components. We extend this concept in order to differentiate between the execution of a complete workflow and a simple process. The former is described by the class *WFExecution* the latter by the *Invocation* of a *ComponentInstance*. During an invocation the relationships between the input and output *Data* and the set of *ComponentParameters* values used by the specific instance are established. The *StateCollection* contains references to the *Data* involved into stateful operations. It provides a more precise representation of the derivations involving the data exchanged between instances, as well as stateful and intermediate data generated and preserved within a single instance. Finally, in S-PROV we combine system-level provenance with contextual information and metadata that

are relevant to the users' interests and to the workflow's application domain, making provenance useful for the scientists, as well as for the workflow developer and system experts.
S-PROV is available as ontology[3] and is implemented in the MongoDB document-store adopting a representation which offers performant query capabilities. This is achieved by applying techniques such as denormalisation and a flexible indexing strategy for the metadata terms. We proceed now describing the queries and use cases characterising the DARE Lineage Services.

3 The Data-Lineage Service

This section describes the current release of the DARE lineage services managing the data-lineage information captured by the DARE platform, in combination with the *dispel4py* workflow system. Our activities focused on consolidating the methods documentation and deployment of the API. The methods of the API are built on top of the provenance model (S-PROV). The source code is available as part of the S-ProvFlow master repository in the DARE GitLab[4] and the service is deployed in the DARE test-bed. Its specification follows the OpenAPI standard and is accessible to the public[5]. It can be queried adopting standard OpenAPI clients:
https://petstore.swagger.io/?url=https://testbed.project-dare.eu/prov/swagger/#/
or with the dedicated interactive lineage exploration tools:
https://testbed.project-dare.eu/sprovflow-viewer/html/view.jsp.

We proceed with introducing the different lineage use cases envisaged by the DARE platform and how these are implemented by the API. The single methods are illustrated in Table 1-3. We will refer to the methods' number in these tables in the description of the use cases.

Acquisition: Lineage information should be stored at runtime in order to allow users to monitor the execution of their methods. Thus, as the computation progresses. method (1) of Table 1 is used by the workflow application to send regular updates about the ongoing execution. Lineage documents are ingested in JSON format. Each document describes the event associated with the production of new output from one of the processes of the workflow. It contains timestamps, details about the generating process, such as the workflow component, the location of the execution and data derivations. The latter capture the *wasDerivedFrom* relationships between output and input data. Input are referenced by their *id*, while output data is described in terms of its metadata and its location, whether the workflow produced a materialised resource such as a file or an entry in a database. The acquisition methods have been extended with a new *import* functionality (4). This allow to ingest provenance produced by workflows represented and executed in CWL (Common Workflow Language), thereby producing lineage information in CWLProv format[6].

Detailed Monitoring: the execution of a workflow can be monitored at different levels of detail. From the high-level classification of the components introduced by the users, grouped in semantic clusters, to the invocation of the single components' instances. This multi-level representation is supported by

[3] https://gitlab.com/project-dare/s-ProvFlow/blob/master/resources/s-prov-o.owl
[4] https://gitlab.com/project-dare/s-ProvFlow/tree/master/provenance-api
[5] https://testbed.project-dare.eu/prov/swagger/
[6] https://zenodo.org/record/1208478#.XOKep6bYVSw

the underlying S-PROV model shown in Figure 1. This allows us to perform queries that obtain automatically complete processing and attribution information. Details such as location, amount of data produced, execution time and software agents are immediately collected and aggregated without joins to follow references. The method of the API offering such functionality is number (6) of Table 2. We will make use of this numbering schema to reference the API's methods throughout the document.

Search for Runs and Data: the validation and traceability query methods of the API, referred as (5), (10), (11), (16) in Table 2 perform searches on concepts and terms defined by the S-PROV model and vocabulary, and on the terms associated with the properties of the *DataGranules* and *ComponentParameters* (see Figure 1). These are used in combination with their values-ranges to search for data and workflows' executions.

Lineage Queries: the methods of the API referred as (12) (13) in Table 2, allow users to navigate the data derivation graph (*derivedData*, *wasDerivedFrom*) bidirectionally. This is used to build a visual and interactive representation of the trace and requires to specify how much depth should be extracted from the lineage trace of the data. Another method that combines graph traversals at a configurable depth with queries on metadata values-ranges is the *filterByAncestor* (11). It receives a list of data ids and applies a filter excluding those whose ancestors' properties do not match the query parameters.

Aggregations and Summaries: the API provides two high-level summary methods to extract comprehensive information. One of the methods (15) covers processing dynamics, such as data transfer between the components and their concrete instances, indicating additional details, such as the computational nodes and execution modes, depending on the chosen enactment. Another method instead (16) reveals collaborative dynamics, such as data-reuse between people, workflows and infrastructures. These are built interactively by specifying properties of the data produced by the users' runs. Capturing and visualising the reuse of workflows outputs across different methods is relevant to trace the long tail of Science, where the data life cycle is affected by the combined interactions between methods that improving the original data or create derivative products on a potentially large time scale.

Table1: S-ProvFlow API Methods **Provenance acquisition**

Provenance acquisition	
(1) workflowexecutions/insert	Bulk insert of bundle or lineage documents in JSON format
(2) workflowexecutions/<id>/edit	Update of the description of a workflow execution. Users can improve this information in free text.
(3) workflowexecutions/<id>/delete	Delete a workflow execution trace, including its bundle and all its lineage documents.
(4) workflowexecutions/import	Import lineage traces from other workflow systems and maps them to S-PROV allowing their exploration through the S-ProvFlow tools. The current implementation supports the import of traces in CWLProv.

Table 2: S-ProvFlow API Methods **Monitoring, validation and lineage queries**

Monitoring, validation and lineage queries	
(5) workflowexecutions(/<id> | ?<query string>)	Extract documents from the bundle collection by the id of a *WFExecution* or according to a query string which may include usernames, type of the workflow, the components the run *wasAssociatedWith* and their implementations. Data results' metadata and parameters can also be queried by specifying the terms and their min and max values-ranges and data formats. Mode of the search can also be indicated (mode ::= (OR | AND). It will apply to the search upon metadata and parameters' values of each run.
(6) workflowexecutions/<id>/showactivity?<query-string>	Extract detailed information related to the activity related to a *WFExecution* (id). The result-set can be grouped by invocations, instances or components and shows progress, anomalies (such as exceptions or systems' and users' messages), occurrence of changes in the implementation and the rapid availability of accessible data bearing intermediate results. This method can also be used for runtime monitoring.
(7) instances/<id> **(8) invocations/<id>** **(9) components/<id>**	Extract details about a single invocation instance or component by specifying their id. The returning document will indicate the changes that occurred, reporting the instances and the first invocation affected.
(10) data*(/<id>* | *?<query string>)*	Extract *Data* and their *DataGranules*. The data is selected by specifying the id or a *query-string*. Query parameters allow to search by *attribution* to a component or to an implementation, or by the id of the *workflow execution* or of the *invocation that* generated the data. In addition, it is also possible to query by combining more metadata terms with their min and max values-ranges. Mode of the search can also be indicated (mode ::= (OR | AND)).
(11) data/filterByAncestor*?<query string>*	Filter a list of data ids based on the existence of at least one ancestor in their data dependency graph, according to a list of metadata terms and

	their min and max values-ranges. Maximum depth level and mode of the search can also be indicated.
(12) data/<*id*>/derivedData **(13) data/<*id*>/wasDerivedFrom**	Starting from a specific data entity of the data dependency is possible to navigate through the derived data (11) or backwards across the element's data dependencies (12). The number of traversal steps is provided as a parameter (level).
(14) terms*?<query string>*	Return a list of discoverable metadata terms based on their appearance for a list of runIds, usernames, or for the whole provenance archive. Terms are returned indicating their type (when consistently used), min and max values and their number occurrences within the scope of the search.

Table 3: S-ProvFlow API Methods **Comprehensive Summaries**

Comprehensive Summaries	
(15) summaries/workflowexecutions/<*id*>*?<query string*>	Produce a detailed overview of the distribution of the computation, reporting the size of data movements between the workflow components, their instances or invocations across worker nodes, depending on the specified granularity level. Additional information, such as process pid, worker, in- stance or component of the workflow (depending on the level of granularity) can be selectively extracted by assigning these properties to a *groupBy* parameter. This will allow users to perform visual analytics tasks that exploit different level of detail and organisations of the visual data-space.
(16) summaries/collaborative*?<query string*>	Extract information about the reuse and exchange of data between workflow executions based on terms' values- ranges and a group of users. The API method allows for inclusive or exclusive (mode ::= (OR | AND) queries on the terms' values. As above, additional details, such as running infrastructure, type and name of the workflow can be selectively extracted by assigning these properties to a *groupBy* parameter. This will support the generation of grouped views.

3.1 Managing Data Properties to Assist Discovery

Users can search for workflow executions and data elements adopting terms which refer to standard vocabularies, as well as experimental terms introduced by specific application's and researchers' requirements. Searching for workflows' executions allows them to configure the visual tools for the exploration of a specific workflow run. Here they can search for experiment and data or apply filters on a collection of data entities previously retrieved (*i.e.* on the properties of their ancestors (10)). Each data product is described by its metadata and the information about its generating process.
The selection of the terms for the searching or the filtering is assisted through hints. These are suggested among the terms introduced by the user's runs attributed to the S-PROV entities such as *ComponentParameters* and *DataGranule* (Figure 1). The hints are presented to the user by accessing an additional database collection, the terms summaries, that is regularly updated via the incremental analysis of the whole provenance archive. The update is performed offline by a batch job that, by executing two map-reduce processes on the lineage documents and the terms summaries itself, emits and updates statistics for all the terms introduced by the users' experiments.

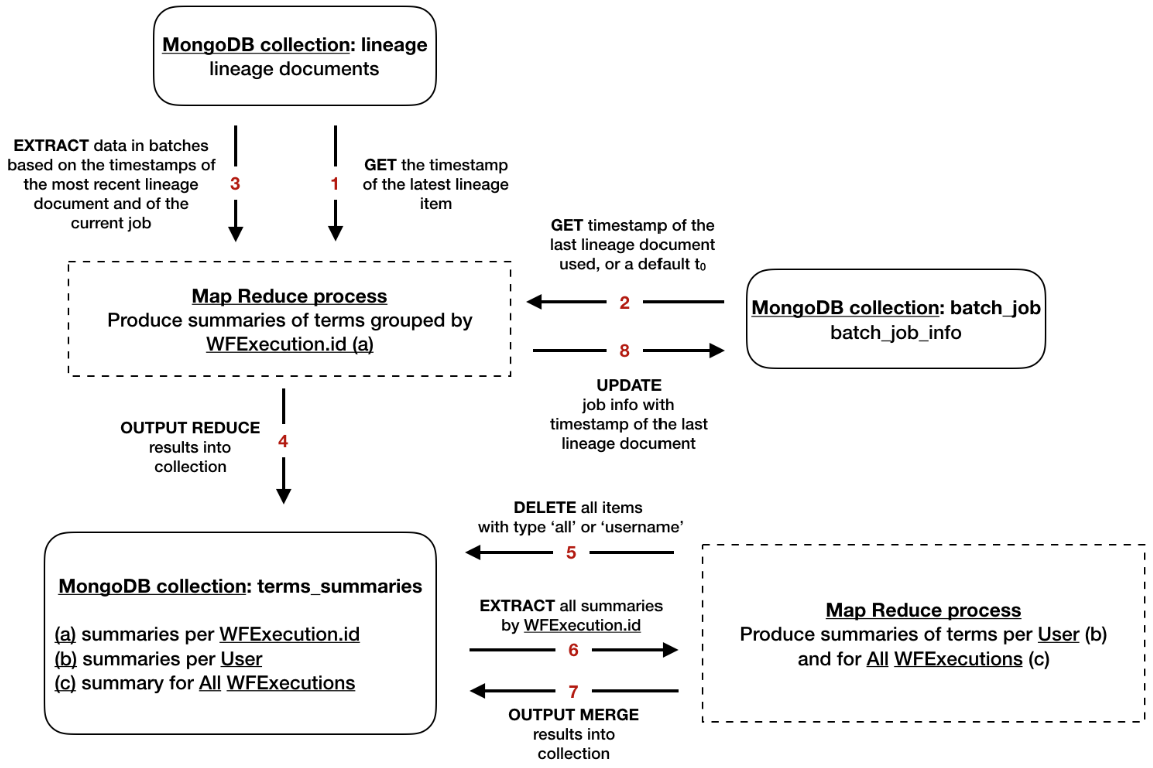
MongoDB collection: lineage
lineage documents
EXTRACT data in batches based on the timestamps of the most recent lineage document and of the current job
3
1
GET the timestamp of the latest lineage item
GET timestamp of the last lineage document used, or a default t0
Map Reduce process
Produce summaries of terms grouped by WFExecution.id (a)
2
8
MongoDB collection: batch_job
batch_job_info
UPDATE job info with timestamp of the last lineage document
OUTPUT REDUCE results into collection
4
DELETE all items with type 'all' or 'username'
5
MongoDB collection: terms_summaries
(a) summaries per WFExecution.id
(b) summaries per User
(c) summary for All WFExecutions
EXTRACT all summaries by WFExecution.id
6
7
OUTPUT MERGE results into collection
Map Reduce process
Produce summaries of terms per User (b) and for All WFExecutions (c)

Figure 2: The image shows the workflow that produces the summaries about the terms that are introduced by the users' run in the provenance archive. The summaries describe the use of the term (metadata or parameters), their type (string or numerical) and statistics (count, max and min values). Three summaries are produced: for single runs (a), users (b) and for the full collection of workflows' executions (c).

The workflow implemented by the batch job is described in Figure 2. This new collection is queried by the *terms* method of the API (14), which returns for each term, the way it is used (metadata or parameter), type, *min* and *max* values, when they have a numerical type, and their number of occurrences within the scope of the search. They may be associated with namespaces prefixes referring to controlled vocabularies or new and experimental.

4 Conclusions and Future Work

The combination of consistent provenance streams with powerful tools supporting interactive access, delivers a smooth path between different levels of expertise that requires to explore the workflow's outcome as needed. The methods are exposed by the web API on top of the underlying lineage model S-PROV, that accommodates complex lineage patterns in data-intensive and stateful operations. The model also represents details about the mapping of the abstract workflow to its distributed and concurrent execution.

Through the API, users and interfaces have easy access to common provenance interrogation use cases. We have shown how we postprocess the lineage to offer hints on metadata terms to be used for discovery that might be relevant in the context of many users and experiments. The system deployment model has been improved adopting docker and it is integrated in the DARE test-bed. This allowed the API to be concretely adopted in the WP6 and WP7 trainings, to monitor workflow executions and validate the obtained results according to the specific scientific aims and terminology.
We envisage future work to validate and align the API with the PROV-AQ recommendation for provenance access and query services, thereby improving interoperability, and in the implementation of the API methods and model, improving its integration and linkage to the DARE Knowledge Base and Workflow Registry. Finally, we want to integrate provenance information capturing the interaction of the user that access and customise their computational context and development environments within specific working sessions. Though, different provenance use case suggest the adoption of different technological choices for its performant and usable storage, thereby, the experimentation of polyglot solutions should be motivated by further work in provenance exploitation scenarios.