1 Accessing provenance information

1.1 State of ProvDAL at Paris meeting, July 2017

ProvDAL is a service the interface of which is organized around one main parameter, the "ID" of an entity (obs_publisher_did of an ObsDataSet for example) or activity. The response is given in one of the following formats: PROV-N, PROV-JSON, PROV-XML, PROV-VOTABLE. Additional parameters can complete ID to refine the query: FORMAT allows to choose the output format. BACKWARD gives the number of relations that shall be tracked in backward direction, i.e. along the provenance history. Its value is either 0, a positive integer or ALL. If this parameter is omitted, the default is ALL, wich returns the complete provenance history. The optional parameter FORWARD defines the number of forward relations; it's also either a positive integer or ALL, but default is 0. That means if neither FORWARD nor BACKWARD are specified, then the complete provenance history is returned.

The ID parameter is allowed more than once in order to retrieve several data set provenance details at the same time. An example request could look like this:

```
{provdal-base-url}?ID=rave:dr4&BACKWARD=1&FORMAT=PROV-JSON
```

Each of the provenance relation has a direction, BACKWARD follows these directions whereas FORWARD follows the relations in reverse direction, independent of the relation type. This is easier to implement, but has the (for a user unexpected) side effect that e.g. agent relations are only retrieved when using BACKWARD, but never with FORWARD. Similarly for membership (hadStep, hadMember) relations: members of a collection or activityFlow are retrieved only in BACKWARD direction, and collections or activityFlows that contain an entity or activity are only found in FORWARD direction. In order to provide a more user-friendly interface with less surprising behaviour, we define three more request parameters: EXPAND_AGENT, EXPAND_COLLECTION and EXPAND_ACTIVITYFLOW. They take TRUE or FALSE as arguments. If they are set to TRUE, the relations with agents, collections and activityFlows will be included in any case, independent of the direction in which the provenance graph is retrieved.

TODO:
Draw a provenance graph picture here with different relation types and arrows for direction.

TODO:
Implementations need to show if this is really the best way.

TODO:
If EXPAND_AGENT=TRUE: include all agent relations, but if EXPAND_AGENT=FALSE, then use default behaviour? Or do not include any of the agent relations? Which one would it be?

A ProvDAL service MUST implement the parameters ID, BACKWARD and FORMAT; the remaining parameters are optional. If a service does not implement the optional parameters, but they appear in the request, then the service should return with an error.

Table 2 summarizes the parameters for such a ProvDAL service interface.

Parameter	**Requirement**	**Value/options**	**Default**	**Description**
ID	required	qualified ID	–	a valid qualified identifier for an entity or activity (can occur multiple times)
BACKWARD	required	0,1,2,..., ALL	ALL	number of relations to be followed backwards or `ALL` for everything
FORWARD	optional	0,1,2,..., ALL	0	number of relations to be followed forward or `ALL` for everything
FORMAT	required	PROV-N, PROV-JSON, PROV-XML, PROV-VOTABLE	?	serialisation format of the response
EXPAND_AGENT	optional	TRUE or FALSE	TRUE	include agent relations in any case
EXPAND_COLLECTION	optional	TRUE or FALSE	TRUE	include relations with collections in any case
EXPAND_ACTIVITYFLOW	optional	TRUE or FALSE	TRUE	include relations with activityFlows in any case

Table 1: ProvDAL request parameters

1.2 New proposal after Paris meeting, after first implementation

ProvDAL is a simple data access layer interface (see DALI specification of the VO, **?**) that can be implemented by a web service to serve provenance information to a client. The client sends GET request to the basic URL endpoint (`{provdal-base-url}`) of a ProvDAL service, providing at least the main parameter **ID**, the (unique, qualified) identifier of an entity (obs_publisher_did of an ObsDataSet for example), activity or an agent. This parameter can occur more than once in a request in order to retrieve provenance details for several activities, datasets or agents at the same time. Here are two simple example requests:

```
{provdal-base-url}?ID=rave:dr4
{provdal-base-url}?ID=rave:dr4&ID=rave:act_irafReduction
```

Additional parameters can complete the request to refine the query. They are described in the next paragraphs and summarized in Table 2.

RESPONSEFORMAT The format of the response can be defined using the RESPONSEFORMAT parameter. Its value is one of the provenance serialization formats: PROV-N, PROV-JSON, PROV-XML, PROV-VOTABLE.

DEPTH The DEPTH parameter gives the number of relations that shall be tracked along the provenance history – independent of the type of relation. Its value is either 0, a positive integer or ALL. If this parameter is omitted, the default is 1, which returns all relations and nodes that can be reached by following 1 relation. If `DEPTH=ALL` is requested, the server should return the complete provenance history that the service has stored for the given entity, activity or agent. Services may restrict the returned data by redirecting `DEPTH=ALL` to `DEPTH={maxdepth}`, where `{maxdepth}` is an integer defining the maximum depth number that the server allows.

If description classes (EntityDescription etc.) are used, we expect that the descriptions are retrieved together with each corresponding main class. For example, for each entity node also return the entityDescription – either by adding the description attributes to the entity in the serialized response, by adding a link to a URL with entityDescription information or by returning a direct serialization of the description class along with a link from the entity to the entityDescription.

Note that the relations *wasDerivedFrom* and *wasInformedBy* are “short-cuts” in a provenance graph. Thus for e.g. `DEPTH=2` more progenitors of an entity may be reached via *wasDerivedFrom* relations than via the “long path” along the corresponding *used* and *wasGeneratedBy* relations (see e.g. progenitor entity E1 in Figure 1). (A better solution for the future may be to use 1/2*DEPTH for walking along these short-cut relations, but we don’t want to make ProvDAL more complex for now.)

Parameter	Values	Description
ID	qualified ID	a valid qualified identifier for an entity, activity or agent (can occur multiple times)
DEPTH	0,<u>1</u>,2,..., ALL	number of relations to be followed or `ALL` for everything, independent of the relation type
RESPONSEFORMAT	PROV-N, <u>PROV-JSON</u>, PROV-XML, PROV-VOTABLE	serialisation format of the response
DIRECTION	<u>BACK</u>, FORTH	BACK = track the provenance history, FORTH = explore the results of activities and where entities have been used
MEMBERS	true (1) or <u>false</u> (0)	if true/1, retrieve and track members of collections
STEPS	true (1) or <u>false</u> (0)	if true/1, retrieve and track steps of activityFlows
AGENT	true (1) or <u>false</u> (0)	if true/1, explore all relations for agents, i.e. find out what an agent is responsible for
MODEL	<u>IVOA</u> or W3C	compatibility of the serialization to the IVOA or W3C provenance data model

Table 2: ProvDAL request parameters. Options that are **required** to be implemented by ProvDAL services are marked with bold face. <u>Default</u> values are underlined. The parameter names are case-insensitive, but the parameter values are not.

DIRECTION For services which allow tracking the provenance information forward, e.g. in order to check for which activities an entity was used, the optional parameter DIRECTION can be set to FORTH. Its default value is BACK. This only influences the direction in which the used, wasGeneratedBy, wasDerivedFrom and wasInfluencedBy relations are followed. Any other relations are tracked according to the behaviour specified below, independent of the DIRECTION value.

Figure 1 shows an example provenance graph with different relations and nodes. Only the relations marked by solid lines are influenced by the DIRECTION parameter. A ProvDAL GET request with `ID=E6` and `DEPTH=2` returns only the highlighted nodes and relations (thick lines) by default.

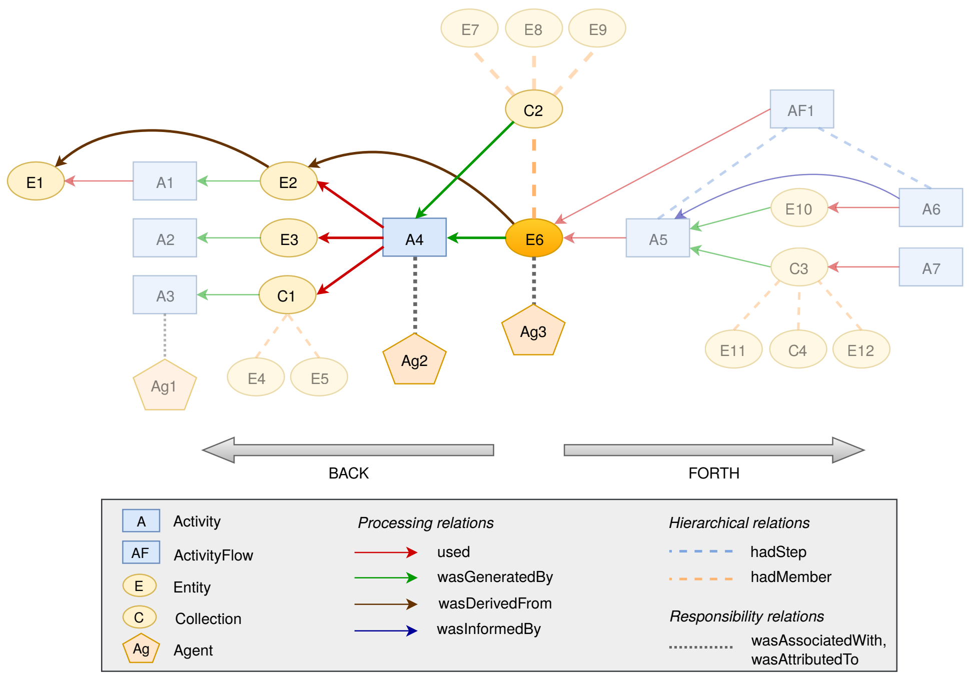

Figure 1: An example provenance graph, highlighting the objects and relations returned from a ProvDAL service with `ID=E6` and `DEPTH=2`. The BACK and FORTH values for DIRECTION are only important for the processing relations (solid lines). Hierarchial (dashed) and responsibility (dotted) relations are only followed "upwards" (to collection/activityFlow) and towards agents by default (unless the optional parameters MEMBERS, STEPS and/or AGENT are set to true.

MEMBERS, STEPS The provenance data model defines the hierarchical relations *hadMember* for entity collections and *hadStep* for activityFlows. If a node belongs to a collection or activityFlow, these relations shall be returned as well, independent of the specified tracking direction. If someone is interested in more details and wants to follow the *members* of an entity collection or the *steps* of an activityFlow, these can be included by setting the optional parameter MEMBERS or STEPS to true, respectively. The default is false. As detailed in DALI (**?**), the values 1 and 0 are equivalent to true and false.

AGENT By default, it is recommended to stop any further tracking at an agent node, unless an additional optional parameter AGENT is set to true. Note that this means that the request for any agent will always return just the agent node itself and nothing else, unless AGENT=true is used. An example request if one wants to know which entities and activities an agent has influenced could look like this:

`{provdal-base-url}?ID=org:rave&AGENT=true&DEPTH=1`.
`DEPTH=1` is used here in order to avoid following the found entities and activities any further (can be omitted, since this is the default for DEPTH).

MODEL If a provenance service supports it, the MODEL parameter is used to distinguish between serializations compatible with the IVOA or the W3C provenance data model. The default value is IVOA.

A ProvDAL service MUST implement the parameters ID, DEPTH and RESPONSEFORMAT; the remaining parameters are optional. If a service does not implement the optional parameters, but they appear in the request, then the service should return with an error. Please note that according to the DALI specification (?), the parameter names are case-insensitive, but the parameter values are not. E.g. `direcion=FORTH` is allowed, but `DIRECTION=forth` may not work.

1.2.1 ProvDAL example use cases

Here are a few example use cases for ProvDAL in order to show its usefulness for astronomical datasets.

- The RAVE DR4 release contains a main table with stellar properties for each observation of a star. Given the RAVE observation ID, retrieve all the processing steps for this specific observation result:

  ```
  {provdal-base-url}?ID=rave:20121220_0752m38_089&DEPTH=ALL
  ```

  The result will not only contain the processing steps (activities), but also entities and agents. The information that the user is actually interested in can be filtered out by a client application (e.g. using the voprov python package). If a W3C tool shall be used, e.g. when the result shall be loaded to ProvStore[1] for further processing, one needs to retrieve the information in a W3C compatible way like this:

  ```
  {provdal-base-url}?ID=rave:20121220_0752m38_089&DEPTH=ALL&MODEL=W3C
  ```

- Get the direct progenitor of an entity:

  ```
  {provdal-base-url}?ID=rave:20121220_0752m38_089&DEPTH=1
  ```

  If this request only returns a collection and no "backwards" information about progenitors, then one needs to track the collection further, i.e. repeat the request for the collection entity.

[1]https://provenance.ecs.soton.ac.uk/store/

- Get all datasets that were derived from a specific data file in the CTA pipeline:

  ```
  {provdal-base-url}?ID=cta:df1&DEPTH=ALL&DIRECTION=FORTH
  ```

ProvDAL is meant to be used to just retrieve parts of a provenance graph from a provenance web service. It cannot be used to explore the provenance graphs and filter the information based on specific properties, e.g. the creationTime of an entity or a parameter value for an activity. For such cases, a ProvTAP service can be used – which is described in the next section.