International
Virtual
Observatory
Alliance

ProvenanceDM Implementation Note

Version 0.1

IVOA Note 2018-04-10

Working group
DM
This version
http://www.ivoa.net/documents/ProvImplementationNote/20180410
Latest version
http://www.ivoa.net/documents/ProvImplementationNote
Previous versions
no previous versions yet
Author(s)
Kristin Riebe, Mathieu Servillat, François Bonnarel, Mireille Louys, Michèle Sanguillon, IVOA Data Model Working Group
Editor(s)
Kristin Riebe, Mathieu Servillat

Abstract

TODO:
This document is still very preliminary and may change any time.

This document collects use cases, guidelines and implementation notes for the IVOA Provenance Data Model as described in Riebe and Servillat et al. (2017). It contains details on the used classes and attributes for each use case, design decision, problems and open issues. It is separated from the standard document for IVOA Provenance, since it contains non-normative information.

Status of this document

This is an IVOA Note expressing suggestions from and opinions of the authors. It is intended to share best practices, possible approaches, or other perspectives on interoperability with the Virtual Observatory. It should not be referenced or otherwise interpreted as a standard specification.

A list of current IVOA Recommendations and other technical documents can be found at http://www.ivoa.net/documents/.

Contents

1 How to use the data model

The IVOA Provenance Data Model has been developed along with its implementations from different projects. We gather here some tips to implement and use the model for a specific project.

Before using the model We noticed that the simple knowledge of what is provenance information is important for the conception of all projects. Before using or not ProvenanceDM and associated services, it is good practice to locate and collect information on the activities, entities and agents that will be manipulated, and be sure that this information is not lost along the way. For example, a script may use intermediate files (such as calibration files for observations) that may not be tracked by the system.

Define unique identifiers It would be convenient if each data object or even each file gets a unique id that can be referenced. The W3C provenance model requires ids for entities, activities and agents, and they have to be qualified strings, i.e. containing a namespace. For example, an activity in the RAVE-pipeline could have the id '`rave:radialvelocity_pipeline_20160901`'. Using a namespace for each project for these ids will help to make them unique. IVOIDs, DOI's or ORCIDs are potentially good candidates for unique identifiers.

Use of the description classes One may only use the core data model without the description classes if they are not needed for the project. In that case, it is recommended to merge the attributes of the description classes into the main classes (*Entity*/*Activity*/*Agent*, and if needed *Parameter*). In the same way, when serializing the provenance information, the description classes can be merged to the main classes, which is needed to produce W3C compliant provenance files.

Add project specific attributes to your entities, activities and agents We proposed generic attributes for the different classes, but there are probably project specific attributes that need to be added. It is also required at this level to disentangle *Entity* properties and *EntityDescription* properties: everything that you may know about an entity before its creation, belongs to the *EntityDescription* (e.g. file `format`, `content_type`, `dataproduct_type`, `category`, ...).

Group common features for entities and/or activities If inside the project, the different entities that will be manipulated are already defined precisely, as well as the activities producing them, then the description classes are probably of interest and will help reducing the redundancy in the provenance information stored.

Create ActivityDescription files A model using description classes to define templates for activities and entities has an advantage for normalization: the common processes could be described once and for all at some place and then be reused when recording provenance information for certain entities

and activities. If description classes are relevant for a project, it may be sufficient to store the descriptions directly as *ActivityDescription* files. Those can be created and centralized before implementing a provenance database or service.

Link to an Authentication System We proposed to add optional attributes to the *Agent* class (email, phone, address). Some projects may include an authentication system with a user directory. In that case a link should be kept between the *Agent* identifiers and the ones used in the authentication system when it is relevant, and information should be synchronized. However, there may be some agents that are not defined in the authentication system, so the information may not be easy to merge.

Adding additional metadata to an existing Entity It could happen that after creating entities and storing their provenance, additional metadata is needed for those entities. For example, an ObsCoreDM description may have to be added to some entities when they are made available to the public. In that case, the entity will receive an external identifier (e.g. a `publisher_did`), which should thus be associated to the entity provenance identifier (Entity.id) in a relation table. This allows to match existing Ids between different data models metadata descriptions as highlighted in the mapping table in the standard document.

2 `voprov` Python package

The `voprov`[1] package is an open source Python library derived from the `prov` Python library (MIT license) developed by Trung Dong Huynh (University of Southampton).

The `prov` package implements the W3C Provenance Data Model. It offers to describe the provenance and provides different output formats of the serialized data: PROV-N, PROV-JSON, PROV-XML and to build diagrams in the following graphic formats: PDF, PNG, SVG.

The `voprov` library allows users to describe the provenance of their data according to the IVOA Provenance Data Model. It allows the description of flows of activities (pipelines) with eventually the composition of the different steps. It provides the VOTable serialization in PROV-VOTable format.

This library is currently used in the context of the POLLUX database which hosts synthetic stellar spectra. The provenance files are created from the non normalized information found in the Pollux header files. The serialization is proposed in three levels of detail and in different formats: on

[1] https://github.com/sanguillon/voprov

one hand in PROV-N, PROV-JSON, PROV-XML and PROV-VOTable formats and on the other hand in PDF, PNG and SVG graphic formats. The VO user or the VO tool is informed of the existence of the provenance in a DataLink entry of the Simple Spectral Access protocol (SSAP) response which gives information on how to retrieve a given provenance file.

The following is an example of a python program which describes the provenance and generates the different formats of serialization.

```
import sys
from prov.model import ProvDocument
from prov.dot import prov_to_dot
import pdb
try:
    provdoc = ProvDocument()
    provdoc.add_namespace('prov', 'http://www.w3.org/ns/prov#')
    provdoc.add_namespace('voprov', 'http://www.ivoa.net/documents/dm/provdm/voprov/')
    provdoc.add_namespace('ivo', 'http://www.ivoa.net/documents/rer/ivo/')
    provdoc.add_namespace('hips', 'http://cds.u-strasbg.fr/data/')
    provdoc.entity('ivo://CDS/P/DSS2color#RGB_NGC6946',\
        {'voprov:name':'RGB DSS2 image for NGC 6946', \
         'voprov:annotation':'PNG RGB image for galaxy NGC 6946'})
    provdoc.entity('ivo://CDS/P/DSS2/POSSII#POSSII.J-DSS2.143',\
        {'voprov:name':'POSSII Blue Survey DSS2 NGC6946', \
         'voprov:annotation':'Blue POSSII Schmidt survey around NGC 6946'})
    provdoc.entity('ivo://CDS/P/DSS2/POSSII#POSSII.F-DSS2.143',\
        {'voprov:name':'POSSII Red Survey DSS2 NGC6946', \
         'voprov:annotation':'Red POSSII Schmidt survey around NGC 6946'})
    provdoc.entity('ivo://CDS/P/DSS2/POSSII#POSSII.N-DSS2.143',\
        {'voprov:name':'POSSII Infra Red Survey DSS2 NGC6946', \
         'voprov:annotation':'Infrared POSSII Schmidt survey around NGC 6946'})
    provdoc.activity('hips:AlaRGB1', '2017-04-18T17:28:00', '2017-04-19T17:29:00', \
          {'voprov:name':'Aladin RGB 1',\
           'voprov:annotation': 'Aladin RGB image generation for NGC 6946',\
           'voprov:desc_name': 'Aladin RGB image generation algorithm', \
           'voprov:desc_type': 'RGBencoding', \
           'voprov:desc_doculink':'http://aladin.u-strasbg.fr/aladin.gml'})
    provdoc.used('hips:AlaRGB1', 'ivo://CDS/P/DSS2/POSSII#POSSII.J-DSS2.143')
    provdoc.used('hips:AlaRGB1', 'ivo://CDS/P/DSS2/POSSII#POSSII.F-DSS2.143')
    provdoc.used('hips:AlaRGB1', 'ivo://CDS/P/DSS2/POSSII#POSSII.N-DSS2.143')
    provdoc.wasGeneratedBy('ivo://CDS/P/DSS2color#RGB_NGC6946', 'hips:AlaRGB1', '')
    f_out = open('ex1.json','w')
    f_out.write(provdoc.serialize(indent=2))
    f_out.close()
    f_out = open('ex1.provn','w')
    f_out.write(provdoc.get_provn())
    f_out.close()
    provdoc.serialize(format='xml', destination='ex1.xml')
    f_out = open('ex1.votable','w')
    f_out.write(provdoc.serialize(format='votable',indent=2))
    f_out.close()
except Exception, e:
        print("Error while writing the file : %s !" %str(e))
        sys.exit(1)
```

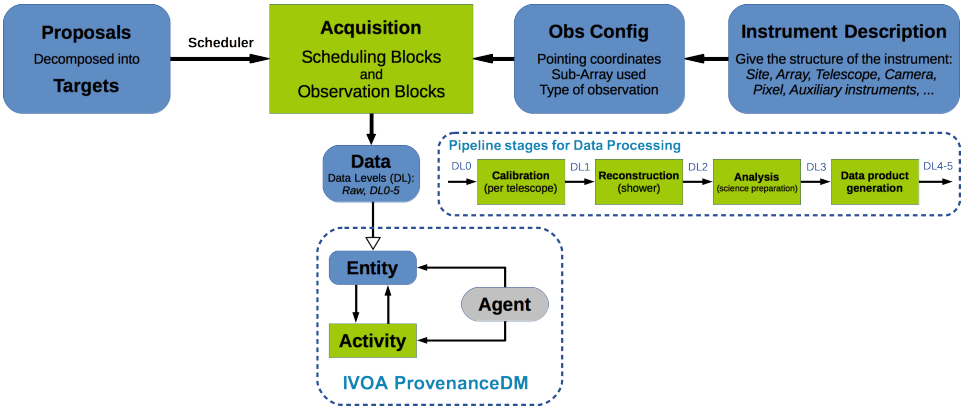

Figure 1: CTA high level data model structure with Pipeline stages and connection to IVOA ProvenanceDM.

3 Provenance for CTA

The Cherenkov Telescope Array (CTA) is the next generation ground-based very high energy gamma-ray instrument. It will provide a deep insight into the non-thermal high-energy universe. Contrary to previous Cherenkov experiments, it will serve as an open observatory providing data to a wide astrophysics community, with the requirement to propose self-described data products to users that may be unaware of the Cherenkov astronomy specificities. The proposed structure of the metadata is presented in Figure 1.

Cherenkov telescopes indirectly detect gamma-rays by observing the flashes of Cherenkov light emitted by particle cascades initiated when the gamma-rays interact with nuclei in the atmosphere. The main difficulty is that charged cosmic rays also produce such cascades in the atmosphere, which represent an enormous background compared to genuine gamma-ray-induced cascades. Monte Carlo simulations of the shower development and Cherenkov light emission and detection, corresponding to many different observing conditions, are used to model the response of the detectors. With an array of such detectors the shower is observed from several points and, working backwards, one can figure out the origin, energy and time of the incident particle. The main stages of the CTA Pipeline are presented inside Figure 1. Because of this complexity in the detection process, provenance information of data products is necessary to the user to perform a correct scientific analysis.

Provenance concepts are relevant for different aspects of CTA:

- Data diffusion: the diffused data products have to contain all the relevant context information with the assumptions made as well as a description of the methods and algorithms used during the data processing.

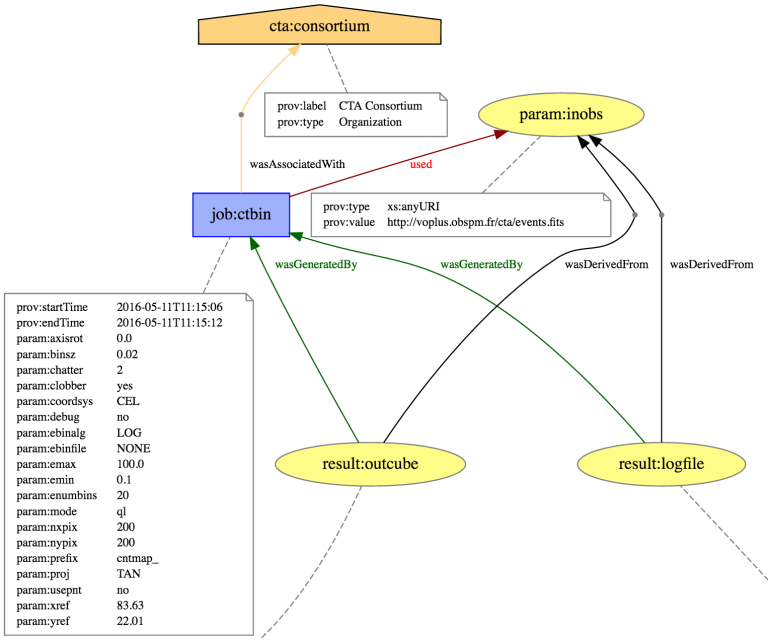

Figure 2: Provenance description of a CTA analysis step.

- Pipeline: the CTA Observatory must ensure that data processing is traceable and reproducible.
- Instrument Configuration: the characteristics of the instrument at a given time have to be available and traceable (hardware changes, measurements of e.g. a reflectivity curve of a mirror, ...)

We tested the tracking of Provenance information during the data analysis using the Python `prov` package inside OPUS[2] (Observatoire de Paris UWS System), a job control system developed at PADC (Paris Astronomical Data Centre). This system has been used to run CTA analysis tools and provides a description of the Provenance in the PROV-XML or PROV-JSON serializations, as well as a graph visualization (see Figure 2).

The CTA Pipeline contains a specific Provenance class dedicated to the collection of provenance information after each processing step. This information is returned as an output file for now.

4 Provenance for the POLLUX database

POLLUX is a stellar spectra database proposing access to high resolution synthetic spectra computed using the best available models of atmosphere (CMFGEN, ATLAS and MARCS), performant spectral synthesis

[2] https://github.com/ParisAstronomicalDataCentre/OPUS

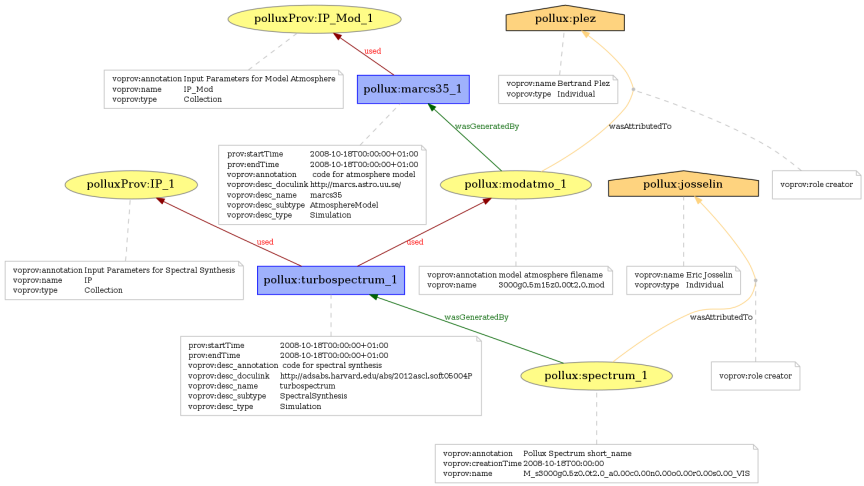

Figure 3: Example for provenance metadata for a POLLUX spectrum, produced from a W3C compatible format (e.g. using `prov:startTime` instead of `voprov:startTime`).

codes (CMF_FLUX, SYNSPEC and TURBOSPECTRUM) and atomic line lists from VALD database and specific molecular line lists for cool stars.

Currently the provenance information is given to the astronomer in the header of the spectra files (depending on the format: FITS, ASCII, XML, VOTable, ...) but in a non-normalized description format.

The implementation of the provenance concepts in a standardized format allows users on one hand to benefit from tools to create, visualize and transform to another format the description of the provenance of these spectra and on a second hand to select data depending on provenance criteria. An example visualization of (W3C compatible) provenance metadata for a POLLUX spectrum is given in Figure 3.

5 Provenance of HiPS datasets

HiPS (Fernique and Allen et al., 2015) is an IVOA recommendation for a new hierarchical organization of pixel data, allowing smooth browsing of all-sky data collections in astronomy. It is based on the HEALPix tesselation of the sky into equal-area cells for a given HEALPix order, where each cell is given an image tile. Adaptative resolution is achieved by a hierarchy of tiles at increasing order. Sorting and organization is based on a tree of nested directories, each of which is associated with a tile. In the processing chain, HiPS can be seen as a kind of “legacy level” for observational data.

A HiPS dataset can be generated either by Aladin in “hipsgen” mode or by other software. The processing distinguishes 3 main different methods for

estimating cell values with parameters: FIRST (nearest neighbour), MEAN, and MEDIAN of the neighbouring pixels. Up to 50 parameters can help to tune the processing, among them the highest resolution's HEALPix order, the sky background value to be substracted, the border width or the mask to apply to original images in order to avoid including bad areas, etc.

An example of provenance metadata for a HiPS collection generated from a collection of SERC Schmidt plates scanned by CAI-Observatoire de Paris with the MAMA facility and serialized in PROV-N format is given at http://wiki.ivoa.net/twiki/bin/view/IVOA/ProvSerialisationExample/HiPS-prov-provn.txt, the corresponding VOTable-format is available at http://wiki.ivoa.net/twiki/bin/view/IVOA/ProvSerialisationExample/HiPS-prov-vot.xml.

Here is an excerpt of the corresponding PROV-N serialization:

```
prefix ivo <http://www.ivoa.net/documents/rer/ivo/>
prefix ex <http://www.example.com/provenance/>
prefix voprov <http://www.ivoa.net/documents/dm/provdm/voprov/>
prefix ds <http://www.ivoa.net/documents/dm/datasetdm/>
prefix hips <http://www.ivoa.net/documents/apps/hips/>
Entity
( ivo://CDS/P/MAMA/ESO-R,
[ voprov:name = "ESO-R MAMA HIPS at CDS",
voprov:type= "voprov:entity",
voprov:annotation = "HiPS version of ESO Schmidt survey digitized by Mama and processed by CDS",
voprov:doculink = "http://cds.u-strasbg.fr/hips/documentation.html#structure",
ds:calibLlevel = 3,
ds:dataproduct_type = "voprov:hips_pixels",
ds:access_reference = "http://CDS/P/MAMA/ESO-R", // as defined in ObsCore and Dataset DM
hips:HiPS_properties = "http://cds.u-strasbg.fr/hips/p/mama/eso-r/properties.txt"] )
// Relationship
WasAttributedTo(ivo://CDS/P/MAMA/ESO-R, ivo://cds, voprov:role= "voprov:creator")
Agent
(ivo://cds,
[ voprov:name= "CDS",
voprov:email = "question@astro.unistra.fr",
voprov:type = "Organisation" ])
WasGeneratedBy  (ivo://CDS/P/MAMA/ESO-R, EHG1, -)
Activity
(EHG1,
[ voprov:name = "ESO HiPS generation 1",
voprov:startTime = "2016-07-18",
voprov:endTime = "2016-07-20",
voprov:annotation = "Final generation activity of HiPS for ESO Mama survey",
voprov:activityDescription = "HipsgenM"] )
ActivityDescription
(HipsgenM,
[ voprov:name = "HiPSgen_Mean",
voprov:type = "data encoding",
voprov:subtype= "HiPSgen",
voprov:doculink = "http://cds.u-strasbg.fr/HiPSGEN-Documentation"])
WasAssociatedWith( EHG1, Buga, voprov:role="voprov:operator")
WasAssociatedWith(EHG1, ivo://CDS, voprov:role="voprov:creator")
WasAttributedTo(( ivo://CDS/P/MAMA/ESO-R, buga, voprov:role= "voprov:operator")
```

6 Provenance for the RAVE project

The RAVE (RAdial Velocity Experiment) [3] is a survey that observed the spectra of half a million stars from the southern hemisphere. In a pipeline of several steps the data were calibrated, reduced and stellar properties were determined, which were then released in the form of star catalogues.

Providing provenance information, e.g. from which spectrum and fibre a dataset was coming from and which steps were involved in processing the data, can help scientists to understand the data, the restrictions and judge the data quality. It would also be useful to be able to compare if, how and why the derived data for some stars have changed between different releases.

6.1 Django web application

General information We set up a web application by implementing the Provenance DM classes along with provenance metadata from the RAVE project using the Django Python Web Framework[4] and storing the metadata information in an SQLite database. In such a Django application, each class of the Provenance Data Model can be implemented as a class (a Django "model") in the webapp. Django automatically creates a database table for each class, with one column for each defined class attribute. Figure 4 shows the ProvenanceDM classes which were implemented for this use case and exist as tables in the (SQLite) database of the webapp.

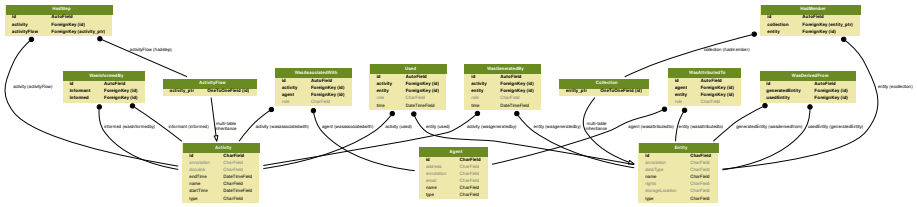

Figure 4: This figure shows the classes that were actually implemented in the RAVE web application along with their attributes (light grey = optional) and relations to each other. This figure was extracted directly from the web application using `manage.py graph_models`. The full resolution image can be found in the volute repository, http://volute.g-vo.org/svn/trunk/projects/dm/provenance/implementation-note/provenance-rave-django-models.pdf.

We split the web application into two parts: a reusable package called `django-prov_vo`[5] and the main web application. The `django-prov_vo`

[3] http://www.rave-survey.org/

[4] https://www.djangoproject.com/

[5] https://github.com/kristinriebe/djangp-prov_vo/

package contains an implementation of the provenance classes, interfaces to access the data and the functionality to serializa the complex provenance information into different formats.

Features Using this Django web application, it is possible to rather quickly setup a provenance web service with the following features:

- List all instances of a class (e.g. all activities or all entities)
- Show details for a single object (e.g. all the properties for a selected activity)
- ProvDAL access to retrieve provenance information for a given entity, activity or agent identifier
- Serialisation of provenance information into different formats (PROV-N, PROV-JSON), also in W3C-compatible form
- Visualization of provenance information with Javascript

REST API The first two points, listing instances and showing details for each ProvenanceDM class are implemented as a REST API, using the Django RESTFramework[6] module. Here are some example URLs for this:

- `{base-url}/api/activities/`: list of all activity instances
- `{base-url}/api/activities/?format=json`: same, in simple json format
- `{base-url}/api/activities/rave:act_classificationPipeline/`: details for the activity with ID `rave:act_classificationPipeline`
- `{base-url}/api/used/`: list of all used relations

Identifiers For each activity, agent or entity, an identifier (ID) is defined using the "rave" namespace defined for this project. This is useful for ensuring unique IDs. A namespace is also required for constructing qualified identifiers for W3C compatible serialisations.

6.2 Implementation details

The following sections are more technical and describe with more details some of the issues that arose during the implementation and their solutions.

[6] http://www.django-rest-framework.org/

6.2.1 Tracking provenance

Provenance for an entity, activity or agent can be tracked backwards or even forward via the different relations using database queries. Here we used a straight-forward approach, without further optimisation, by implementing three recursive functions: one for finding an *activity* and following its relations, one for tracking an *entity* and one for tracking an *agent*. They search for the desired activity/entity/agent in any of the corresponding relations (e.g. for activity, check used, wasGeneratedBy, wasInformedBy, hadStep), extract the activity, entity or agent the desired activity/entity/agent is linked with and call the corresponding function for tracking it further (recursively).

6.2.2 ProvDAL

The IVOA ProvenanceDM specification (Riebe and Servillat et al., 2017) describes a method to retrieve provenance information for a specified entity, activity or agent via a ProvDAL service interface. We have implemented it for this web application, using `provdal/` as endpoint with following request parameters:

- `ID`: a valid qualified identifier for an entity or activity (can occur multiple times)
- `DEPTH`: a positive integer (including 0) for following the provenance graph this number of relations or `ALL` for everything
- `RESPONSEFORMAT`: format of the serialized response, currently only `PROV-JSON` and `PROV-N` are implemented
- `DIRECTION`: direction of the provenance tracking, either `BACK` or `FORTH`. This only affects the way how *used*, *wasGeneratedBy*, *wasDerivedFrom* and *wasInformedBy* are followed.
- `MEMBERS`, `STEPS`, `AGENT`: additional flags that can be `true` or `false`.
- `MODEL`: extra parameter for choosing a serialization compliant to `IVOA` ProvenanceDM or `W3C` ProvDM

This allows to retrieve provenance descriptions in the desired format and compliant to W3C or including the IVOA ProvenanceDM extensions.

We additionally implemented following non-standard extension:

- `RESPONSEFORMAT=GRAPH` retrieves a html page with an interactive visualisation of the provenance description

A typical request looks like this:

```
{base-url}/provdal/?ID=rave:20121220_0752m38_089&DEPTH=1&
    RESPONSEFORMAT=PROV-N&MODEL=IVOA
```

This generates a serialised description of the entity with ID `rave:20121220_0752m38_089` and its direct relations, in PROV-N format:

```
document
prefix prov <http://www.w3.org/ns/prov#>
prefix voprov <http://www.ivoa.net/documents/ProvenanceDM/ns/voprov/>
prefix custom <http://www.ivoa.net/documents/ProvenanceDM/ns/custom/>
prefix xsd <http://www.w3.org/2000/10/XMLSchema#>
prefix rave <http://www.rave-survey.org/prov/>
prefix org <http://www.ivoa.net/documents/ProvenanceDM/ns/org/>
prefix vo <http://www.ivoa.net/documents/ProvenanceDM/ns/vo>
activity(rave:act_dataextraction, 2003-04-11T14:42:16Z, 2012-05-27T17:05:00Z,
    [voprov:name="Data Extraction", voprov:type="obs:Extraction",
    voprov:annotation="RAVE observations, data release 4",
    voprov:doculink="RAVE DR4: Kordopatis et al. 2013, AJ, 146, Issue 5,
    article id. 134, pp.421-451"])
entity(rave:20121220_0752m38_089, [voprov:name="20121220_0752m38_089",
    voprov:annotation="Properties of observed star derived for this
    observation in RAVE DR4", voprov:rights="voprov:public",
    custom:dataType="databaseRow",
    custom:storageLocation="https://www.rave-surve.org/query/"])
entity(rave:DR4_RAVEDR4, [voprov:name="RAVEDR4 main",
    voprov:annotation="The main RAVE DR4 table", voprov:rights="voprov:public",
    custom:dataType="databaseTable",
    custom:storageLocation="https://www.rave-survey.org/project/documentation/dr4/rave_dr4/"])
wasGeneratedBy(rave:20121220_0752m38_089, rave:act_dataextraction, -)
hadMember(rave:DR4_RAVEDR4, rave:20121220_0752m38_089)
endDocumentdocument
```

We implemented a web form[7] as frontend for the ProvDAL interface, in which all the parameters can be set. The form does not support asking for multiple IDs at once for now.

Content negotiation with HTTP Accept We also check, if the HTTP accept header is compatible with the requested response format. If it is not, a 406 Not Accepted error is returned. The following table shows the accept types that we considered compatible with the implemented serialization format. If either an unsupported REPSONSEFORMAT value or accept header is requested, the HTTP error 415 Not Supported is returned.

[7] https://escience.aip.de/provenance-rave/prov_vo/provdal

RESPONSEFORMAT	HTTP accept
PROV-N	text/plain text/* */*
PROV-JSON	application/json application/* */*

6.2.3 Collection

For RAVE, each pipeline step operates on a number of files which can be summarized as a collection. E.g. the processing step "IRAF Reduction" produces a set of files with reduced spectra, which are summarized as "IRAF Reduced dataset", a collection. The individual files are members of this collection (via *hadMember* relationship).

A *Collection* is derived from the Entity-class and thus inherits the same attributes and relations. Since we did not need any additional attributes for collections, we did not make use of the extra *Collection* class, and only marked entities that are collections by setting their type-attribute to "collection" (like it is done in W3C serialisations). Thus, when looping over a list of entities, it is also faster to check if they are a collection (only need to check an attribute and not check if an entity occurs in the list of collections as well).

6.2.4 Serializations

We implemented serializers for each class, which directly map the used classes and attributes into the required format, adding the voprov-namespace to each attribute. This is useful for processing the provenance information with ProvenanceDM-aware VO-tools. However, according to the IVOA Provenance DM requirements, it must also be possible to serialize the provenance information into at least one of the W3C formats. Thus we also implemented W3C variants of the serializers. These map the additonal classes and attributes from ProvenanceDM to a valid W3C representation.

Here's a list of the main changes/mappings for converting from the IVOA model to W3C serialisation:

- namespace `voprov` → `prov` for those attributes that are the same in W3C (e.g. ID, role, startTime, endTime)
- attribute `voprov:name` → `prov:label`
- attribute `voprov:annotation` → `prov:description`

- attribute `prov:role` is not allowed in W3C's *wasAttributedTo*, thus use `voprov:role`
- *hadMember* has no ID and no optional attributes in W3C
- *Collection* → *Entity* with `prov:type = prov:collection`

W3C serialization of *ActivityFlow* The classes *ActivityFlow* and *HadStep* do not exist in W3C. When a W3C compliant serialization of a provenance description including an activityFlow is needed, these classes have to be expressed with existing W3C classes:

- Represent each activityFlow as an activity; additional attribute `voprov:votype = 'voprov:activityFlow'`
- Represent each hadStep relation as a wasInfluencedBy relation; additional attribute `voprov:votype = 'voprov:hadStep'`

For example, an activityFlow and related activities in IVOA ProvenanceDM, PROV-N notation may look like this:

```
activityFlow(rave:act_pipeline, -, -, [voprov:name="RAVE Pipeline"])
activity(rave:act_sparvPipeline, -, -, [voprov:name="SPARV-Pipeline"])
activity(rave:act_irafReduction, -, -, [voprov:name="IRAF Reduction"])
hadStep(rave:act_pipeline, rave:act_irafReduction)
hadStep(rave:act_pipeline, rave:act_sparvPipeline)
wasInformedBy(rave:act_sparvPipeline, rave:act_irafReduction)
```

and the corresponding W3C serialisation would look like this:

```
activity(rave:act_pipeline, -, -, [voprov:votype="voprov:activityFlow", voprov:name="RAVE Pipeline"])
activity(rave:act_sparvPipeline, -, -, [voprov:name="SPARV-Pipeline"])
activity(rave:act_irafReduction, -, -, [voprov:name="IRAF Reduction"])
wasInfluencedBy(rave:act_pipeline, rave:act_irafReduction, [voprov:votype="voprov:hadStep"])
wasInfluencedBy(rave:act_pipeline, rave:act_sparvPipeline, [voprov:votype="voprov:hadStep"])
wasInformedBy(rave:act_sparvPipeline, rave:act_irafReduction)
```

This way, an activityFlow can be expressed with W3C compliant terms, and it is possible to convert from this representation to an IVOA-compliant version including activityFlow and hadStep explicitly. Note also, that there exists also at least one other attempt to model workflows, e.g. D-PROV[8], which is using extensions to W3C Prov.

6.2.5 VOSI resources

The necessary interfaces `availability` and `capabilities` are implemented in an additional reusable package, `django-vosi`.

[8] https://www.usenix.org/system/files/conference/tapp13/tapp13-final3.pdf

6.2.6 Sources, demo

Please note that this RAVE Provenance web application is still a prototype and may change any time.

- Sources of the web application:
  https://github.com/kristinriebe/provenance-rave
  https://github.com/kristinriebe/django-prov_vo
  https://github.com/kristinriebe/django-vosi
- Live demo version:
  https://escience.aip.de/provenance-rave
  The base-url for the given examples is
  https://escience.aip.de/provenance-rave/prov_vo/
- Javascript example for provenance visualisations:
  https://escience.aip.de/prov/graphs/example.html

7 Provenance from SimDM classes for CosmoSim

etc. etc.

References

Fernique, P., Allen, M., Boch, T., Donaldson, T., Durand, D., Ebisawa, K., Michel, L., Salgado, J. and Stoehr, F. (2015), 'Hips - hierarchical progressive survey version 1.0', IVOA Recommendation.
http://www.ivoa.net/documents/HiPS/20170519/

Riebe, K., Servillat, M., Bonnarel, F., Louys, M., Rothmaier, F., Sanguillon, M. and the IVOA Data Model Working Group (2017), 'IVOA provenance data model', IVOA Working Draft.
http://www.ivoa.net/documents/ProvenanceDM/