International
Virtual
Observatory
Alliance

IVOA Provenance Data Model
Version 1.0

IVOA Working Draft 2017-09-12

Working group
DM
This version
http://www.ivoa.net/documents/ProvenanceDM/20170912
Latest version
http://www.ivoa.net/documents/ProvenanceDM
Previous versions
WD-ProvenanceDM-1.0-20161121.pdf
ProvDM-0.2-20160428.pdf
ProvDM-0.1-20141008.pdf
Author(s)
Kristin Riebe, Mathieu Servillat, François Bonnarel, Mireille Louys, Florian Rothmaier, Michèle Sanguillon, IVOA Data Model Working Group
Editor(s)
Kristin Riebe, Mathieu Servillat

Abstract

This document describes how provenance information for astronomical datasets can be modeled, stored and exchanged within the astronomical community in a standardized way. We follow the definition of provenance as proposed by the W3C[1], i.e. that provenance is information about entities, activities, and people involved in producing a piece of data or thing, which can be used to form assessments about its quality, reliability or trustworthiness. Such provenance information in astronomy is important to enable any scientist to trace back the origin of a dataset (e.g. an image, spectrum, catalog or single points in a spectral energy distribution diagram or a light curve), learn about the people and organizations involved in a project and assess the quality of the dataset as well as the usefulness of the dataset for her own scientific work.

Status of This Document

This is an IVOA Working Draft for review by IVOA members and other interested parties. It is a draft document and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use IVOA Working Drafts as reference materials or to cite them as other than "work in progress".

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

Contents

Acknowledgments

This document has been developed in part with support from the German Astrophysical Virtual Observatory, funded by BMBF Bewilligungsnummer 05A14BAD and 05A08VHA. The Provenance Working Group acknowledges support from the ASTERICS Project, funded by the European Commission (project 653477).

Thanks for fruitful discussions to (in alphabetical order): Markus Demleitner, Harry Enke, Jochen Klar, Gerard Lemson, Markus Nullmeier and Adrian Partl.

Conformance-related definitions

The words "MUST", "SHALL", "SHOULD", "MAY", "RECOMMENDED", and "OPTIONAL" (in upper or lower case) used in this document are to be interpreted as described in IETF standard, Bradner (1997).

The *Virtual Observatory (VO)* is a general term for a collection of federated resources that can be used to conduct astronomical research, education, and outreach. The International Virtual Observatory Alliance (IVOA) is a global collaboration of separately funded projects to develop standards and infrastructure that enable VO applications.

1 Introduction

In this document, we discuss a draft of an IVOA standard data model for describing the provenance of astronomical data. We follow the definition of provenance as proposed by the W3C (Belhajjame and B'Far et al., 2013), i.e. that provenance is "information about entities, activities, and people involved in producing a piece of data or thing, which can be used to form assessments about its quality, reliability or trustworthiness".

In astronomy, such entities are generally datasets composed of VOTables, FITS files, database tables or files containing values (spectra, lightcurves), logs, parameters, etc. The activities correspond to processes like an observation, a simulation, or processing steps (image stacking, object extraction, etc.). The people involved can be individual persons (observer, publisher...), groups or organisations. An example for activities, entities and agents as they can be discovered backwards in time is given in Figure 1.

The currently discussed Provenance Data Model is sufficiently abstract that its core pattern could be applied to any kind of process using either observation or simulation data. It could also be used to describe the workflow for observation proposals or the publication of scientific articles based on (astronomical) data. However, here we focus on astronomical data. The

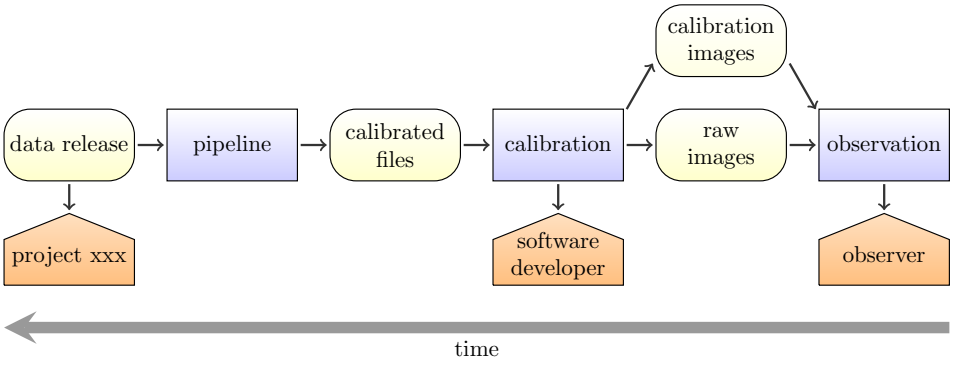

Figure 1: An example graph of provenance discovery. Starting with a released dataset (left), the involved activities (blue boxes), progenitor entities (yellow rounded boxes) and responsible agents (orange pentagons) are discovered.

links between the Provenance Data Model and other IVOA data models will be discussed in Section 3. We note here that the provenance of simulated data is already covered by the Simulation Data Model (SimDM, Lemson and Wozniak et al., 2012). Therefore we also give a mapping between SimDM and the Provenance Data Model in Section 3.

1.1 Goal of the provenance model

The goal of this Provenance Data Model is to describe how provenance information can be modeled, stored and exchanged. Its scope is mainly modeling of the flow of data, of the relations between data, and of processing steps.

Characteristics of observation activities such as ambient conditions and instrument characteristics can be associated to provenance information. during the execution of processing activities (computer structure, nodes, operating system used...) can also be connected to provenance information. However, they will not be modeled here explicitly. This additional information can be included in the form of data or entities linked to those activities, or as attributes of activities (see also Section 2.2.7 for parameters of activities).

In general, the model shall capture information in a machine-readable way that would enable a scientist who has no prior knowledge about a dataset to get more background information. This will help the scientist to decide if the dataset is adequate for her research goal, assess its quality and get enough information to be able to trace back its history as far as required or possible.

Provenance information may be recorded in minute detail or by using

coarser elements, depending on the intended usage and the desired level of detail for a specific project that records provenance. This granularity depends on the needs of the project and the intended usage when implementing a system to track provenance information.

The following list is a collection of tasks which the Provenance Data Model should help to solve. They are flagged with [S] for problems which are more interesting for the end user of datasets (usually a scientist) and with [P] for tasks that are probably more important for data producers and publishers. More specific use cases in the astronomy domain for different types of datasets and workflows along with example implementations are given in Section 6.

A: Tracking the production history [S]

Find out which steps were taken to produce a dataset and list the methods/tools/software that was involved. Track the history back to the raw data files/raw images, show the workflow (backwards search) or return a list of progenitor datasets.
Examples:

- Is an image from catalogue xxx already calibrated? What about dark field subtraction? Were foreground stars removed? Which technique was used?
- Is the background noise of atmospheric muons still present in my neutrino data sample?

We do not go as far as to consider easy reproducibility as a use case – this would be too ambitious. But at least the major steps undertaken to create a piece of data should be recoverable.

B: Attribution and contact information [S]

Find the people involved in the production of a dataset, the people/organizations/institutes that need to be cited or can be asked for more information.
Examples:

- I want to use an image for my own work – who was involved in creating it? Who do I need to cite or who can I contact to get this information? Is a license attached to the data?
- I have a question about column xxx in a data table. Who can I ask about that?
- Who should be cited or acknowledged if I use this data in my work?

C: Locate error sources [S, P]

Find the location of possible error sources in the generation of a dataset.
Examples:

- I found something strange in an image. Where does the image come from? Which instrument was used, with which characteristics etc.? Was there anything strange noted when the image was taken?
- Which pipeline version was used – the old one with a known bug for treating bright objects or a newer version?
- This light curve doesn't look quite right. How was the photometry determined for each data point?

D: Quality assessment [P]

Judge the quality of an observation, production step or dataset.
Examples:

- Since wrong calibration images may increase the number of artifacts on an image rather than removing them, knowledge about the calibration image set will help to assess the quality of the calibrated image.

E: Search in structured provenance metadata [P, S]

This would allow one to also do a "forward search", i.e. locate derived datasets or outputs, e.g. finding all images produced by a certain processing step or derived from data which were taken by a given facility.
Examples:

- Give me more images that were produced using the same pipeline.
- Give me an overview on all images reduced with the same calibration dataset.
- Are there any more images attributed to this observer?
- Which images of the Crab Nebula are of good quality and were produced within the last 10 years by someone not from ESO or NASA?
- Find all datasets generated using this given algorithm for this given step of the data processing

This task is probably the most challenging. It also includes tracking the history of data items as in A, but we still have listed this task separately, since we may decide that we can't keep this one, but we definitely want A.

1.2 Minimum requirements for provenance

We derived from our goals and use cases the following minimum requirements for the Provenance Data Model:

- Provenance information must be stored in a standard model, with standard serialization formats.
- Provenance information must be machine readable.
- Provenance data model classes and attributes should be linked to other IVOA concepts when relevant (DatasetDM, ObsCoreDM, SimDM, VOTable, UCDs...).
- Provenance information should be serializable into the W3C provenance standard formats (PROV-N, PROV-XML, PROV-JSON) with minimum information loss.
- Provenance metadata must contain information to find immediate progenitor(s) (if existing) for a given entity, i.e. a dataset.
- An entity must point to the activity that generated it (if the activity is recorded).
- Activities must point to input entities (if applicable).
- Activities may point to output entities.
- Provenance information should make it possible to derive the chronological sequence of activities.
- Provenance information can only be given for uniquely identifiable entities, at least inside their domain.
- Released entities should have a main contact.
- It is recommended that all activities and entities have contact information and contain a (short) description or link to a description.

1.3 Role within the VO architecture

The IVOA Provenance Data Model is structuring and adding metadata to trace the original process followed during the data production to provide astronomical data. Even if it borrows the main general concepts defined from the data management science, it binds to the specific context of astronomical metadata description and re-uses or interacts with existing IVOA models. It takes benefits from existing IVOA notations and standards like

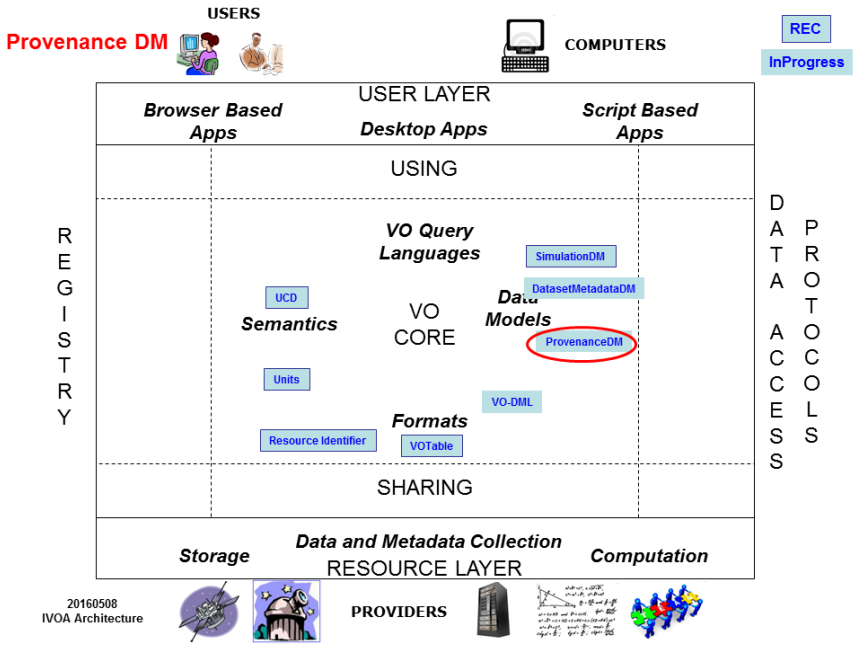

Figure 2: Architecture diagram for the Provenance Data Model. It is based on existing concepts defined in existing IVOA data models, and existing formats and semantics and fully integrated in the IVOA framework

UCD, VOUnits, VO protocols and service design and is planned for a full integration into the VO landscape.

Fig. 2 shows the dependencies of this document with respect to other existing standards.

1.4 Previous efforts

The provenance concept was early introduced by the IVOA within the scope of the Observation Data Model (see IVOA note by IVOA Data Model Working Group, 2005) as a class describing where the data is coming from. A full observation data model dedicated to the specific spectral data was then designed (Spectral Data Model, McDowell and Salgado et al., 2016) as well as a fully generic characterisation data model of the measurement axes of the data (Characterisation Data Model, IVOA Data Model Working Group, 2008) while the progress on the provenance data model was slowing down.

The IVOA Data Model Working Group first gathered various use cases coming from different communities of observational astronomy (optical, radio, X-ray, interferometry). Common motivations for a provenance tracing of the history included: quality assessment, discovery of dataset progenitors and access to metadata necessary for reprocessing. The provenance data

model was then designed as the combination of *Data processing*, *Observing configuration* and *Observation ambient conditions* data model classes. The *Processing class* was embedding a sequence of processing stages which were hooking specific ad hoc details and links to input and output datasets, as well as processing step description. Despite the attempts of UML description of the model and writing of xml serialization examples the IVOA effort failed to provide a workable solution: the scope was probably too ambitious and the technical background too unstable. A compilation of these early developments can be found on the IVOA site (Bonnarel and the IVOA Data Model Working Group, 2016). From 2013 onwards IVOA concentrated on use cases related to processing description and decided to design the model by extending the basic W3C provenance structure, as described in the current specification.

Outside of the astronomical community, the Provenance Challenge series (2006 – 2010), a community effort to achieve inter-operability between different representations of provenance in scientific workflows, resulted in the Open Provenance Model (Moreau and Clifford et al., 2010). Later, the W3C Provenance Working Group was founded and released the W3C Provenance Data Model as Recommendation in 2013 (Belhajjame and B'Far et al., 2013). OPM was designed to be applicable to anything, scientific data as well as cars or immaterial things like decisions. With the W3C model, this becomes more focused on the web. Nevertheless, the core concepts are still in principle the same in both models and very general, so they can be applied to astronomical datasets and workflows as well. The W3C model was taken up by a larger number of applications and tools than OPM, we are therefore basing our modeling efforts on the W3C Provenance data model, making it less abstract and more specific, or extending it where necessary.

The W3C model even already specifies PROV-DM Extensibility points (section 6 in Belhajjame and B'Far et al. 2013) for extending the core model. This allows one to specify additional roles and types to each entity, agent or relation using the attributes `prov:type` and `prov:role`. By specifying the allowed values for the IVOA model, we can adjust the model to our needs while still being compliant to W3C.

2 The provenance data model

In this section, we describe the currently discussed Provenance Data Model. We start with an UML class diagram, explain the core elements and then give in the following sections more details for each class and relation.

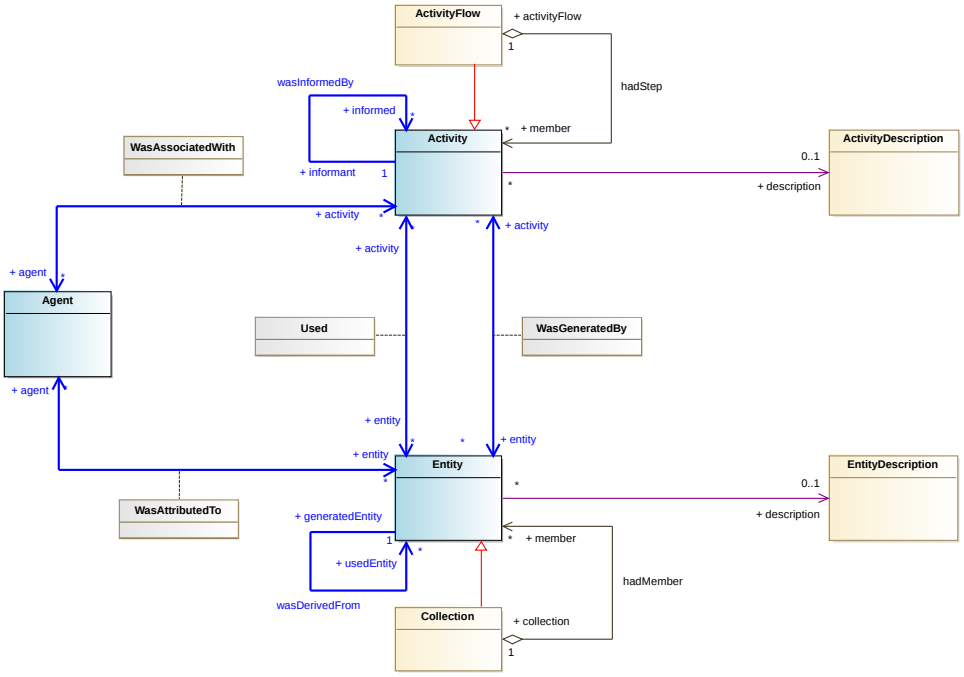

Figure 3: Overview of the classes for the Provenance Data Model in a conceptual class diagram. The blue classes are core elements. There appear a number of many-to-many relationships with attached association classes (grey) which can contain additional attributes.

2.1 Overview: Conceptional UML class diagram and introduction to core classes

Figure 3 shows the conceptional UML diagram for an IVOA Provenance Data Model. The core elements of the Provenance Data Model are *Entity*, *Activity* and *Agent*. We chose for these elements the same names as were used in the Provenance Data Model of the World Wide Web Consortium (W3C, Belhajjame and B'Far et al. 2013), which defines a very abstract pattern that can be reused here. Here are the core classes with a short description and some examples:

- *Entity:* a thing at a certain state
  examples: data products like images, catalogs, parameter files, calibration data, instrument characteristics

- *Activity:* an action/process or a series of actions, occurs over a period of time, performed on or caused by entities, usually results in new entities
  examples: data acquisition like observation, simulation; regridding, fusion, calibration steps, reconstruction

- *Agent:* executes/controls an activity, is responsible for an activity or an entity
  examples: telescope astronomer, pipeline operator, principal investigator, software engineer, project helpdesk

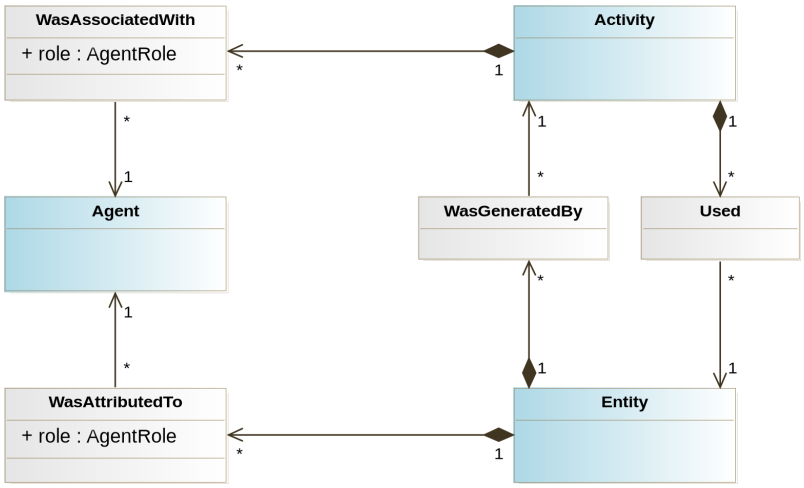

Figure 4: The main core classes and relations of the Provenance Data Model, which also occur in the W3C model.

These core classes along with their relations to each other are provided in Figure 4. We use the following relation classes to specify the mapping between the three core classes. The relation names were again chosen to match the W3C model names:

- *WasGeneratedBy:* a new entity is generated by an activity
  (entity "image m31.fits" wasGeneratedBy activity "observation")
- *Used:* an entity is used by an activity
  (activity "calibration" used entities "calibration data", "raw images")
- *WasAssociatedWith:* agents have responsibility for an activity
  (agent "observer Max Smith" wasAssociatedWith activity "observation")
- *WasAttributedTo:* an entity can be attributed to an agent
  (entity "image m31.fits" wasAttributedTo "M31 observation campaign")

Note that the relations appear as extra classes (and thus boxes in the diagrams, instead of just having annotated relations), because they can have additional attributes – when mapping the model to a relational database, these relations would appear as mapping tables.

In the domain of astronomy, certain processes and steps are repeated again and again with different parameters. We therefore separate the descriptions of activities from the actual processes and introduce an additional *ActivityDescription* class (see Figure 3). Likewise, we also apply the same pattern for *Entity* and add an *EntityDescription* class. Defining such descriptions allows them to be reused, which is very useful when performing a series of tasks of the same type, as is typically done in astronomy.

A similar normalization of descriptions of the actual processes and datasets can also be found in the IVOA Simulation Data Model (SimDM, Lemson and Wozniak et al., 2012)), which describes simulation metadata. The SimDM classes *Experiment* and *Protocol* correspond to the Provenance terms *Activity* and *ActivityDescription*.

This separation into two classes may not be needed for each and every project, and everyone is free to choose which classes make sense for his/her use case. When serializing provenance, one can integrate the description side into the other classes, thus producing a W3C compliant provenance description. More details about all these classes and relations are given in the following section.

2.2 Model description

2.2.1 Class diagram and VO-DML compatibility

Figure 5 shows the full class diagram with the association classes for the many-to-many relations modeled more directly as mapping classes. When implementing the model in a relational database, these classes can be represented as individual tables for mapping the relation. We model one of the associations of the many-to-many relationships as composition (full diamond), if the mapping class belongs more strongly to one of its linked classes, e.g. the *Used* relations are strongly dependent on the corresponding *Activities*. The documentation of all classes and an automatically generated figure based on the underlying xmi-description behind this UML diagram is available in the Volute repository at `https://volute.g-vo.org/svn/trunk/projects/dm/vo-dml/models/provenancedm/vo-dml/ProvenanceDM.html`.

This version of the UML diagram is fully VO-DML compliant, i.e. we just used the restricted subset of UML to model Provenance and reused the IVOA datatypes.

2.2.2 Entity and EntityDescription

Entities in astronomy are usually astronomical or astrophysical datasets in the form of images, tables, numbers, etc. But they can also be observation or simulation log files, files containing system information, environment

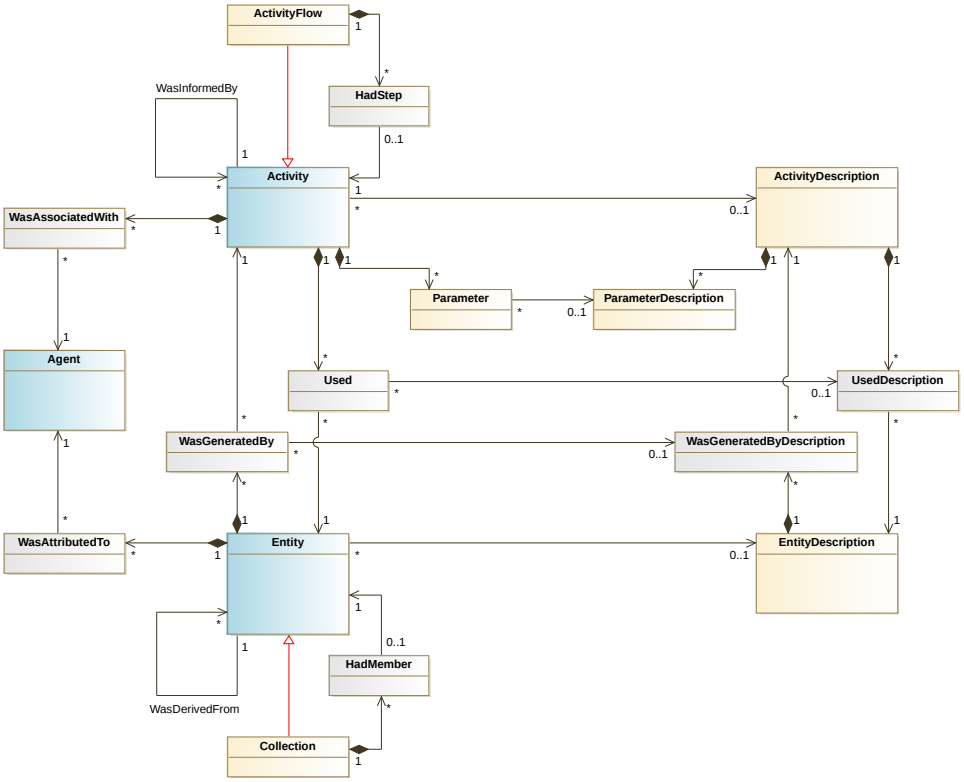

Figure 5: More detailed overview of the classes for the Provenance Data Model. Note that this UML class diagram is more compatible with VO-DML.

variables, names and versions of packages, ambient conditions or, in a wider sense, also observation proposals, scientific articles, or manuals and other documents.

An entity is not restricted to being a file. It can even be just a number in a table, depending on how fine-grained the provenance shall be described.

The VO concept closest to Entity is the notion of "Dataset", which could mean a single table, an image or a collection of them. The Dataset Metadata Model (Bonnarel and Laurino et al., 2015) specifies an "IVOA Dataset" as "a file or files which are considered to be a single deliverable". Most attributes of the *Dataset* class can be mapped directly to attributes of the *Entity* and EntityDescription class, see the mapping table 12 in Section 3.

For entities, we suggest the attributes given in Table 1. If the attribute also exists in the W3C Provenance Data Model, we list its name in the second column.

The difference between entities that are used as input data or output data becomes clear by specifying the relations between the data and activities producing or using these data. More details on this will follow in Section

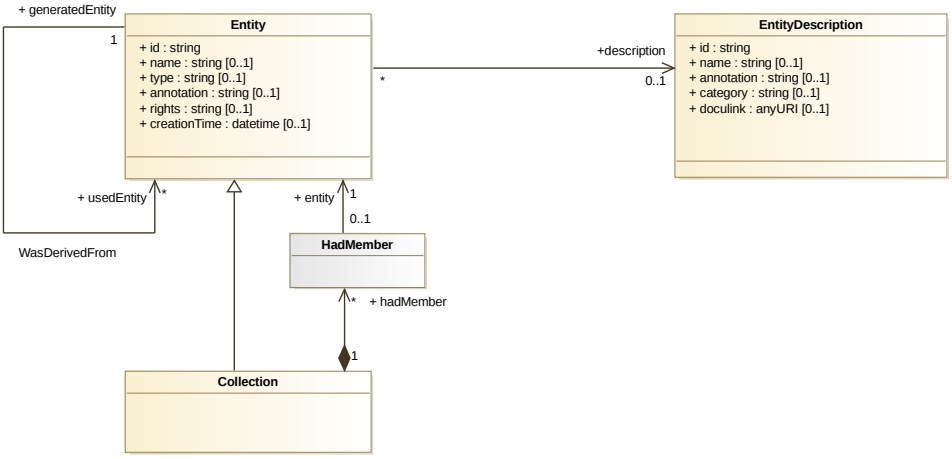

Figure 6: The relation between Entity, EntityDescription and Collection (see Section 2.2.3). Links to the Dataset class from the Dataset Metadata Model are described in Section 3.

2.2.6.

EntityDescription. The types of entities, or datasets in astronomy, can be predefined using a description class *EntityDescription*. This class is meant to store information about an Entity that are known before the Entity instance is created. For example, if we run an activity to create a RGB image from three grey images, we may have a mandatory format for the input and output images before the execution (JPG, PNG, FITS. . .), but we probably cannot know the final size of the image that will be created. Therefore, "format" would be an EntityDescription attribute , while "size" would be an attribute of the Entity instance.

Some of the attributes that describe the content of the data could be derived from the Dataset Metadata Model.

The *EntityDescription* does NOT contain any information about the usage of the data, it tells nothing about them being used as input or output. This is defined only by the relations (and the relation descriptions) between activities and entities (see Section 2.2.6).

The EntityDescription general attributes are summarized in Table 2.

WasDerivedFrom. In Figure 6 there is one more relation that we have not mentioned yet: the *WasDerivedFrom*-relation which links two entities together, borrowed from the W3C model. It is used to express that one entity was derived from another, i.e. it can be used to find one (or more) progenitor(s) of a dataset, without having to look for the activities in between. It can therefore serve as a shortcut.

Entity

Attribute	W3C ProvDM	Data type	Description
id	prov:id	(qualified) string	a unique id for this entity (unique in its realm)
name	prov:label	string	a human-readable name for the entity (to be displayed by clients)
type	prov:type	string	a provenance type, i.e. one of: prov:collection, prov:bundle, prov:plan, prov:entity; not needed for a simple entity
annotation	prov:description	string	text describing the entity in more detail
rights	–	string	access rights for the data, values: public, restricted or internal; can be linked to Curation.Rights from ObsCore/DatasetDM
creationTime	–	datetime	date and time at which the entity was created (e.g. timestamp of a file)

Table 1: Attributes of entities. Mandatory attributes are marked in bold.

The information this relation provides is somewhat redundant, since progenitors for entities can be found through the links to activity and the corresponding descriptions. Nevertheless, we include *WasDerivedFrom* for those cases where an explicit link between an entity and its progenitor is useful (e.g. for speeding up searches for progenitors or if the activity in between is not important).

Note that the *WasDerivedFrom* relation cannot always automatically be infered from following *WasGeneratedBy* and *Used* relations alone: If there is more than one input and more than one output of an activity, it is not clear (without consulting the activityDescription and entity roles in the relation-descriptions) which entity was derived from which. Only by specifying the descriptions and roles accordingly or by adding the a *WasDerivedFrom* relation, this direct derivation becomes known.

2.2.3 Collection

Collections are entities that are grouped together and can be treated as one single entity. From the provenance point of view, they have to have the *same origin*, i.e., they were produced by the same activity (which could

EntityDescription

Attribute	Data type	Description
id	(qualified) string	a unique identifier for this description
name	string	a human-readable name for the entity description
annotation	string	a decriptive text for this kind of entity
category	string	specifies if the entity contains information on logging, system (environment), calibration, simulation, observation, configuration, ...
doculink	url	link to more documentation

Table 2: Attributes of *EntityDescription*. For simple use cases, the description classes may be ignored and its attributes may be used for *Entity* instead.

WasDerivedFrom

Attribute	Data type	Description
id	string	a unique id for this entity (unique in its realm)
generatedEntity	string	foreign key to the entity
usedEntity	string	foreign key to the progenitor, from which the generatedEntity was derived
activity	string	foreign key to the generation activity
generation	string	foreign key to the wasGeneratedBy relation
usage	string	foreign key to the used relation

Table 3: Attributes of the WasDerivedFrom relation. This is the same as used in W3C's ProvDM. Mandatory attributes are marked in bold.

also be the activity of collecting data for a publication or similar). The term "collection" is also used in the Dataset Metadata Model for grouping datasets. As an example, a collection with the name 'RAVE survey' could consist of a number of database tables and spectra files.

The Entity-Collection relation can be modeled using the *Composite* design pattern: Collection is a subclass of Entity, but also an aggregation of 1 to many entities, which could be collections themselves. In order to be compliant to VO-DML, we model the membership-relation explicitly by including a *HadMember* class in our model, which is connected to the *Col-*

lection class via a composition. It may contain an additional role attribute.

Collections are also known in the W3C model, in the same sense as used here. The relation between entity and collection is also called "HadMember" in the W3C model.

An additional class *CollectionDescription* is only needed if it has different attributes than the *EntityDescription*. This class should therefore only be introduced if a use case requires it.

Advantages of collections: Collections can be used to collect entities with the same provenance information together, in order to hide complexity where necessary. They can be used for defining different levels of detail (granularity).

2.2.4 Activity and ActivityDescription

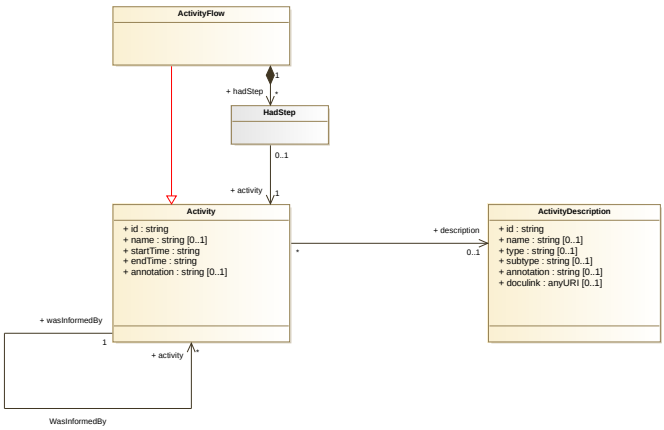

Figure 7: Details for Activity, ActivityDescription and ActivityFlow (see Section 2.2.5).

Activities in astronomy include all steps from obtaining data to the reduction of images and production of new datasets, like image calibration, bias subtraction, image stacking; light curve generation from a number of observations, radial velocity determination from spectra, post-processing steps of simulations etc.

ActivityDescription. The method underlying an activity can be specified by a corresponding *ActivityDescription* class (previously named *Method*, corresponds to the *Protocol* class in SimDM). This could be, for instance, the name of the code used to perform an activity or a more general description of the underlying algorithm or process. An activity is then a concrete case (instance) of using such a method, with a startTime and endTime, and it refers to a corresponding description for further information.

Activity

Attribute	W3C ProvDM	Data type	Description
id	prov:id	(qualified) string	a unique id for this activity (unique in its realm)
name	prov:label	string	a human-readable name (to be displayed by clients)
startTime	prov:startTime	datetime	start of an activity
endTime	prov:endTime	datetime	end of an activity
annotation	prov:description	string	additional explanations for the specific activity instance

Table 4: Attributes of *Activity*, their data types and equivalents in the W3C Provenance Data Model, if existing. Attributes in bold are **mandatory**.

ActivityDescription

Attribute	Data type	Description
id	string	a unique id for this activity description (unique in its realm)
name	string	a human-readable name (to be displayed by clients)
type	string	type of the activity, from a vocabulary or list, e.g. data acquisition (observation or simulation), reduction, calibration, publication
subtype	string	more specific subtype of the activity
annotation	string	additional free text description for the activity
doculink	url	link to further documentation on this process, e.g. a paper, the source code in a version control system etc.

Table 5: Attributes of *ActivityDescription*.

There MUST be exactly zero or one *ActivityDescription* per *Activity*. If steps from a pipeline shall be grouped together, one needs to create a proper *ActivityDescription* for describing all the steps at once. This method can then be refered to by the pipeline-activity.

When serializing the data model, the attributes of the description class may be assigned to the activity in order to produce a W3C compliant serialization (same as with Entity/EntityDescription).

WasInformedBy. The individual steps of a pipeline can be chained together directly, without mentioning the intermediate datasets, using the *WasInformedBy*-relation. This relation can be used as a short-cut, if the exchanged datasets are deemed to be not important enough to be recorded. For grouping activities, also see the next section 2.2.5.

2.2.5 ActivityFlow

TODO:
Link to D-PROV!

For facilitating grouping of activities (and their related entities etc.) we introduce the class *ActivityFlow*. It can be used for hiding and grouping a part of the workflow/pipeline or provenance description, if different levels of granularity are needed. Such pipelines and workflows are very common in astronomical data production and processing. Figure 8 illustrates an example provenance graph in a detailed level (left side) and using the ActivityFlow (right side).

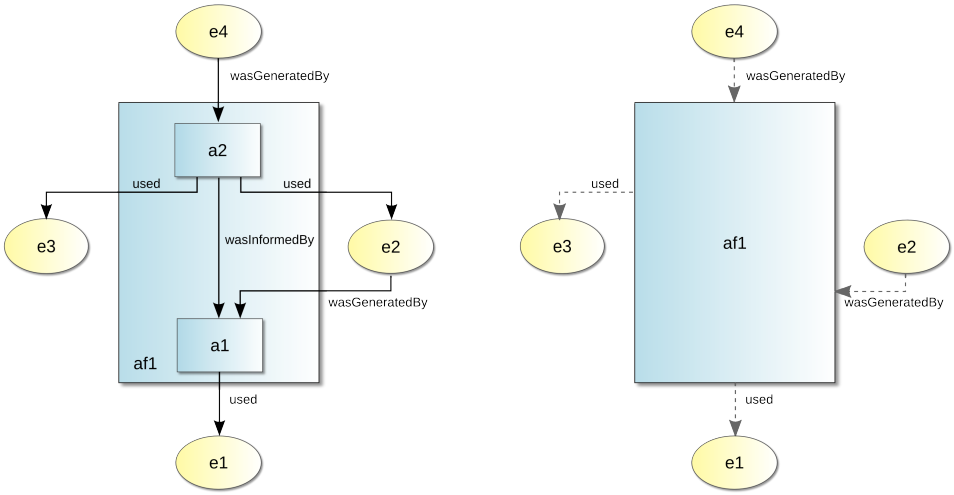

Figure 8: An example provenance graph. The detailed version is shown on the left side. It also shows the shortcut *WasInformedBy* to connect two activities, which could be used if the entity e2 would not be needed anywhere else. An ActivityFlow can be used to "hide" a part of the provenance graph as is shown on the right side. Activities are marked by blue rectangles, entities by yellow ellipses.

We also explored the different ways to describe a set of activities in the W3C provenance model. This model uses *Bundle*, i.e. an entity with type "Bundle", for wrapping a provenance description. Each part of a provenance description can be put into a bundle, and the bundle can then be reused in

other provenance descriptions. W3C's *Plan* is an entity with type "Plan" and is used for describing a set of actions or steps. Both, *Bundle* and *Plan*, are entities and have the attributes and relations of this class (and thus one can define provenance of bundles and plans as well).

But we would like to consider a set of activities as being an *Activity* itself, with all the relations and properties that an activity also has. Therefore we do not reuse W3C's classes for describing workflows and plans, but added the class *ActivityFlow* as an activity composed of activities. The composition is represented by the "hadStep" relation, as is shown in Figure 7.

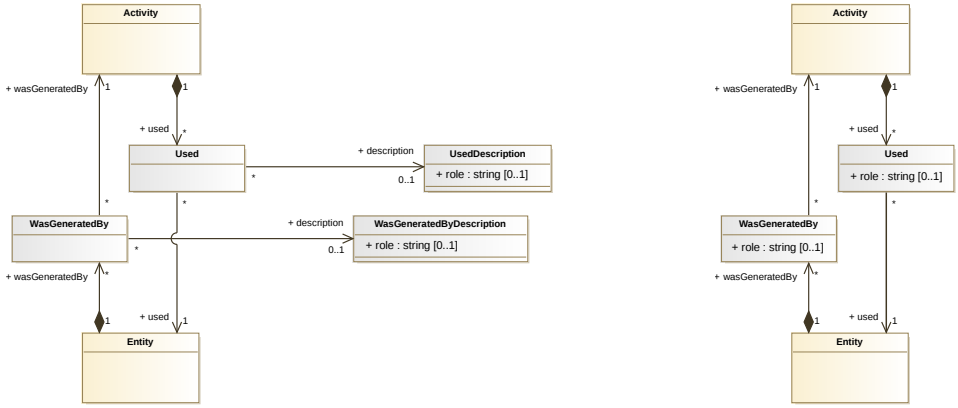

Figure 9: Entity and *Activity* are linked via the *Used* and *WasGeneratedBy* relations. In the left image, the *role* that an entity which was used or generated by an activity played is recorded with the corresponding *UsedDescription* and *WasGeneratedByDescription*, also see Section 2.2.6. If these description classes are not used, the *role* can be used directly as an attribute within the *Used* and *WasGeneratedBy*classes (right image).

2.2.6 Entity-Activity relations

For each data flow it should be possible to clearly identify entities and activities. Each entity is usually a result from an activity, expressed by a link from the entity to its generating activity using the *WasGeneratedBy* relation, and can be used as input for (many) other activities, expressed by the *Used* relation. Thus the information on whether data is used as input or was produced as output of some activity is given by the *relation-types* between activities and entities.

We use two relations, *Used* and *WasGeneratedBy*, instead of just one mapping class with a flag for input/output, because their descriptions and role-attributes can be different.

The *WasGeneratedBy*-relation can have the optional attribute *time* – this is the time, when the generation of the entity is finished. This generation

time corresponds to e.g. *DataID.date* in Dataset Metadata DM.

Compositions and multiplicities In principle, an entity is produced by just one activity. However, by introducing the *ActivityFlow* class for grouping activities together, one entity can now have many wasGeneratedBy-links to activities. One of them must be the actual generation activity, the other activities can only be activityFlows containing this generation-activity. This restriction of having only one "true" generation activity is not explicitly expressed in the current model[2].

The *Used* relation is closely coupled to the *Activity*, so we use a composition here, indicated in Figure 5 by a filled diamond: if an activity is deleted, then the corresponding used relations need to be removed as well. The entities that were used still remain, since they may have been used for other activities as well. We need a multiplicity * between *Used* and *Entity*, because an entity can be used more than once (by different activities).

Similarly, the *WasGeneratedBy* relation is closely coupled with the *Entity* via a composition, since a wasGeneratedBy relation makes no sense without its entity. So if an entity is deleted, then its wasGeneratedBy relation must be deleted as well. There is a multiplicity * between *Activity* and *WasGeneratedBy*, because an activity can generate many entities.

Entity roles Each activity requires specific roles for each input or output entity, thus we store this information with description classes, in the role-attributes for the *UsedDescription* and *WasGeneratedByDescription* relation. For example, an activity for darkframe-subtraction requires two input images. But it is very important to know which of the images is the raw image and which one fulfils the role of dark frame.

The role is in general NOT an attribute for *EntityDescription* or *Entity*, since the same entity (e.g. a specific FITS file containing an image) may play different roles with different activities. If this is not the case, if the image can only play the same role everywhere, only then it is an intrinsic property of the entity and should be stored in the *EntityDescription*.

Some example roles are given in Table 6. Note that these roles don't have to be unique, many datasets may play the same role for a process. For example, many image entities may be used as science-ready-images for an image stacking process.

In order to facilitate interoperability, the possible entity-roles could be defined and described for each activity by the IVOA community, in a vocabulary list or thesaurus.

[2]The reason for this is that we want to keep the model simple and avoid introducing even more classes.

Role	Example entities
configuration	configuration file
auxiliary input	calibration image, dark frame, etc.
main input	raw image, science-ready images
main result	image, cube or spectrum
log	logging output file
red	image used for red channel of a composite activity

Table 6: Examples for entity roles as attributes in the *UsedDescription* and *WasGeneratedByDescription*.

2.2.7 Parameters

The concept of activity configuration, generally a set of parameters that can be configured, is different to the concept of provenance information. However, it is tightly connected. We identify three different ways to link configuration information to an activity:

- Declare a parameter set (or each parameter) as an input entity that is used by the activity.
  This also allows tracking the provenance of the parameter further.
- Define families of activities, each one with fixed attributes.
  I.e. use different subclasses for activities with different fixed attributes.
- Add activity attributes in the form of key-value parameters.

To enable the latter solution, we add a *Parameter* class along with a *ParameterDescription* for describing additional properties of activities. In this solution, Parameters are directly connected to an Activity without complex Entity-Activity relations. Moreover, we can then describe each parameter in the same way as in FIELD and PARAM elements in VOTABLE (Ochsenbein and Williams et al., 2013).

Parameter

Attribute	Data type	Description
id	string	parameter unique identifier
value	(value dependent)	the value of the parameter

Table 7: Attributes of *Parameter*. Attributes in bold are **mandatory**.

For example, observations generally require information on *ambient conditions* as well as *instrument characteristics*. This contextual data associated

ParameterDescription

Attribute	Data type	Description
id	string	parameter unique identifier
name	string	parameter name
annotation	string	additional free text description
datatype	string	datatype
unit	string	physical unit
ucd	string	Unified Content Descriptor, supplying a standardized classification of the physical quantity
utype	string	UType, meant to express the role of the parameter in the context of an external data model
min	number	minimum value
max	number	maximum value
options	list	list of accepted values

Table 8: Attributes of *ParameterDescription*.

with an observation is not directly modelled in the ProvenanceDM. However, this information can be stored as different entities. Alternatively, one could list the instrument characteristics as a set of key-value parameters using the *Parameter* class, so that this information is structured and stored with the provenance information (and can thus be queried simultaneously). In the case of a processing activity that cleans an image with a sigma-clipping method, the input and output images would be entities and the value of the number of sigma for sigma-clipping could be a parameter instead of an entity. We may also want to define a 3-sigma-clipping activity where this parameter is fixed to 3.

2.2.8 Agent

An *Agent* describes someone who is responsible for a certain task or entity, e.g. who pressed a button, ran a script, performed the observation or published a dataset. The agent can be a single person, a group of persons (e.g. MUSE WISE Team), a project (CTA) or an institute. This is also reflected in the IVOA Dataset Metadata Model, where *Party* represents an agent, and it has two types: *Individual* and *Organization*, which are explained in more detail in Table 9 (also see Section 3 for comparison between *Agent* and *Party*). Both agent types are also used in the W3C Provenance Data Model, though *Individual* is called *Person* there. We decided to not include the type *SoftwareAgent* from W3C (yet), since it is not required for our current use

cases. This may change in the future.

AgentType

Class or type	W3C ProvDM	DatasetDM	Comment
Agent	Agent	Party	
Individual	Person	Individual	a person, specified by name, email, address, (though all these parts may change in time)
Organization	Organization	Organization	a publishing house, institute or scientific project

Table 9: Agent class and types of agents/subclasses in this data model, compared to W3C ProvDM and DatasetDM.

Agent

Attribute	W3C ProvDM	Data type	Description
id	prov:id	(qualified) string	unique identifier for an agent
name	prov:name	string	a common name for this agent; e.g. first name and last name; project name, agency name...
type	prov:type	string	type of the agent: either Individual (Person) or Organization

Table 10: Agent attributes

A definition of organizations is given in the IVOA Recommendation on Resource Metadata (Hanisch and the IVOA Resource Registry Working Group et al., 2007), hereafter refered to as RM: "An organisation is [a] specific type of resource that brings people together to pursue participation in VO applications." It also specifies further that scientific projects can be considered as organisations on a finer level: "At a high level, an organisation could be a university, observatory, or government agency. At a finer level, it could be a specific scientific project, space mission, or individual researcher. A provider is an organisation that makes data and/or services available to users over the network."

For each agent a *name* should be specified, a summary of the attributes for *Agent* is given in Table 10. One could also add the optional attributes

address, *phone* and *email* (compare with subclasses of *Party* in Section 3). However, we skip them here in this main class, since an advanced system may use permanent identifiers (e.g. ORCIDs) to identify agents and retrieve their properties from an external system. It would also increase the value of the given information if the (current) affiliation of the agent (and a project leader/group leader) were specified in order to maximize the chance of finding any contact person later on. The contact information is needed in case more information about a certain step in the past of a dataset is required, but also in order to know who was involved and to fulfill our "Attribution" requirement (Section 1.2), so that proper credits are given to the right people/projects.

It is desired to have at least one agent given for each activity (and entity), but it is not enforced. There can also be more than one agent for each activity/entity with different *roles* and one agent can be responsible for more than one activity or entity. This many-to-many relationship is made explicit in our model by adding the two following relation classes:

- wasAssociatedWith: relates an *activity* to an agent
- wasAttributedTo: relates an *entity* to an agent

We adopted here the same naming scheme as was used in W3C ProvDM. Note that the attributed-to-agent for a dataset may be different from the agent that is associated with the activity that created an entity. Someone who is performing a task is not necessarily given full attribution, especially if he acts on behalf of someone else (the project, university, ...).

In order to make it clearer what an agent is useful for, we suggest the possible roles an agent can have (along with descriptions partially taken from RM) in Table 11. For comparison, SimDM contains following roles for their contacts: owner, creator, publisher and contributor. Note that the *Party* class in Dataset and SimDM are very similar to the *Agent* class, which is explained in more detail in Section 3.

This list is *not* complete. We consider providing a vocabulary list for this in a future version of this model, collected from (future) implementations of this model.

AgentRoles

role	type or sub class	Comment
author	Individual	someone who wrote an article, software, proposal
contributor	Individual	someone who contributed to something (but not enough to gain authorship)
editor	Individual	editor of e.g. an article, before publishing
creator	Individual	someone who created a dataset, creators of articles or software are rather called "author"
curator	Individual	someone who checked and corrected a dataset before publishing
publisher	Organization (maybe also Individual?)	organization (publishing house, institute) that published something
observer	Individual	observer at the telescope
operator	Individual	someone performing a given task
coordinator/PI	Individual	someone coordinating/leading a project
funder	Organization	agency or sponsor for a project as in Prov-N
provider	Organization	"an organization that makes data and/or services available to users over the network" (definition from RM)

Table 11: Examples for roles of agents and the typical type of that agent

3 Links to other data models

TODO:
Move this section into appendix?

The Provenance Data Model can be applied without making any links to other IVOA data model classes. For example when the data is not yet published, provenance information can be stored already, but a DatasetDM-description for the data may not yet exist. However, if there are data models implemented for the datasets, then it is very useful to connect the classes and attributes of the other data models with Provenance classes and attributes (if applicable), which we are going to discuss in this Section. These links help to avoid unnecessary repetitions in the metadata of datasets, and also offer the possibility to derive some basic provenance information from existing data model classes automatically.

3.1 Links with Dataset/Obscore Model

Entities and their descriptions in the Provenance Data Model are tightly linked to the *DataSet*-class in the DatasetDM/ObsCore Data Model, as well as to InputDataset and OutputDataSet in the Simulation Data Model (SimDM, Lemson and Wozniak et al., 2012). Table 12 maps classes and attributes from the Dataset Data Model to concepts in the Provenance Data Model.

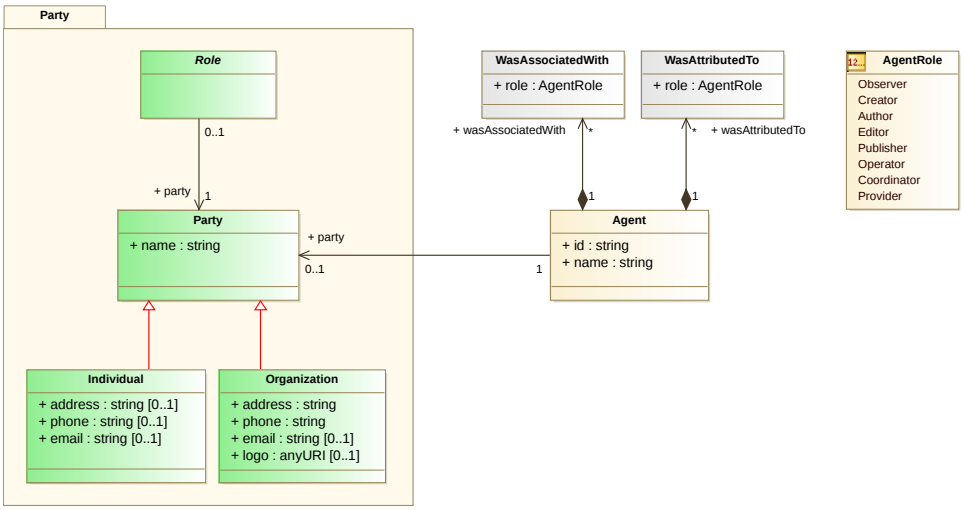

Figure 10: The relations between the *Agent* class within the Provenance Data Model (grey and yellow classes) with classes from the Dataset Metadata Model, party package (green).

The *Agent* class, which is used for defining responsible persons and orga-

nizations in ProvenanceDM, is very similar to the *Party* class in the Dataset Metadata Model (and in SimDM). Its details are depicted in Figure 10. The main difference between *Agent* and *Party* is that *Individual* and *Person* are subclasses in DatasetDM, whereas we just use the same class *Agent* for both and distinguish between them using the *Agent.type* attribute (which can have the value "Individual" or "Organization").

We imagine that services implementing both data models, *Dataset* and *ProvenanceDM* may just use *one* class: either *Agent* or *Party*, enriched with all the necessary (project-specific) attributes. When delivering the data on request, the serialised versions can be adjusted to the corresponding notation. Note that for Provenance queries using a ProvTAP service or for W3C compatible serializations, the name *Agent* for the responsible individuals/organizations is required.

3.2 Links with Simulation Data Model

In SimDM one also encounters a normalization similar to our split-up of descriptions from actual data instances and executions of processes: the SimDM class "experiment" is a type of *Activity* and its general, reusable description is called a "protocol", which can be considered as a type of this model's *ActivityDescription*. More direct mappings between classes and attributes of both models are given in Table 13.

Provenance DM	Dataset DM	Comment
Entity.name	DataID.title	title of the dataset
HadMember.collectionId	DataID.collection	link to the collection to which the dataset belongs
Agent.name	DataID.creator	name of agent
Entity.id	DataID.creatorDID	alternative id for the dataset given by the creator, could be used as Entity.id if no Publisher-DID exists (yet)
WasGeneratedBy.activityId	DataID.ObservationID	identifier to everything describing the observation
WasGeneratedBy.time	DataID.date	date and time when the dataset was completely created
Entity.id	Curation.PublisherDID	unique identifier for the dataset assigned by the publisher
Agent.id	Curation.PublisherID	link to the publisher, i.e. to an Agent with role="publisher"
Agent.name	Curation.Publisher	name of the publisher
Entity.releaseDate	Curation.Date	release date of the dataset
Entity.version	Curation.Version	version of the dataset
Entity.rights	Curation.Rights	access rights to the dataset; one of [...]
Entity.link	Curation.Reference	link to publication
Agent	Curation.Contact	link to Agent with role contact
EntityDescription.dataproduct_type	DataProductType	the type of a dataproduct from Dataset Metadata Model can be used as attribute to entity
EntityDescription.dataproduct_subtype	DataProductSubType	subtype of a dataproduct/entity
EntityDescription.level	ObsDataset.calibLevel	(output) calibration level, integer between 0 and 3

Table 12: Mapping between attributes from Dataset Metadata Model classes to classes in ProvenanceDM

Provenance DM	**Simulation DM**	**Comment**
Activity	Experiment	
Activity.name	Experiment.name	human readable name; name attribute in SimDM is inherited from Resource-class
Activity.endTime	Experiment.executionTime	end time of the execution of an experiment/activity
Activity.activityDescription	Experiment.protocol	reference to the protocol or description class
ActivityDescription	Protocol	
ActivityDescription.name	Protocol.name	human readable name
ActivityDescription.doculink	Protocol.referenceURL	reference to a webpage describing it
Parameter	ParameterSetting	value of an (input) parameter
ParameterDescription	InputParameter	description of an (input) parameter
Agent	Party	responsible person or organization
Agent.name	Party.name	name of the agent
WasAssociatedWith	Contact	
WasAssociatedWith.role	Contact.role	role which the agent/-party had for a certain experiment (activity); SimDM roles contain: `owner`, `creator`, `publisher`, `contributor`
WasAssociatedWith.agent	Contact.party	reference to the agent/-party
Entity	DataObject	a dataset, which can be/refer to a collection

Table 13: Mapping between classes and attributes from ProvenanceDM to classes/attributes in SimDM.

4 Provenance Data Model serialization

4.1 Serializing of the datamodel core

The provenance information as represented in the data model is split in three main concepts that can be searched following many different relations involved between the main 3 classes. The selection of the relations to expose when distributing the provenance information depends on the usage and will be described more extensively in the Use-case sections (6) , the implementation note (Riebe and Servillat et al., 2017) and the links therein.

The serialization documents build up the main components of the client/server dialogs.

To give a very simple example, suppose a client asks for the context of execution for one specified Activity, which computes a simple RGB color composition.

On the server side, exposing the Provenance information for this Activity of name or for an Entity is just exposing the structure of the classes and relation tables and feed them with the related t-upples in the database. On the client side, the content of a VO-Provenance serialisation document can then be explored and represented using graphical interfaces, as inspired by the Provenance Southampton suite or by customized visualization tools.

In the W3C Provenance framework, three descriptions formats are proposed to serialize the Provenance metadata : PROV-N, PROV-JSON, PROV-XML. These are serializations of the W3C provenance data model, a larger set of classes and relations compared to this model but sharing the same core structure. They allow the possibility to add IVOA or *ad hoc* attributes to the basic ones in each class. This way the IVOA models can produce W3C compliant serializations and take benefit of visualizing tools.

Here is an example serialization for an entity being processed by an activity, in PROV-N format:

```
document
  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/>
  entity(ivo://example#Public_NGC6946, [voprov:name="Processed image of NGC 6946"])
  entity(ivo://example#DSS2.143, [voprov:name="Unprocessed image of NGC 6946"])
  activity(ex:Process1, 2017-04-18T17:28:00, 2017-04-19T17:29:00, [voprov:name="Process 1"])
  used(ex:Process1, ivo://example#DSS2.143, -)
  wasGeneratedBy(ivo://example#Public_NGC6946, ex:Process1, 2017-05-05T00:00:00)
endDocument
```

```
{
  "prefix": {
    "ivo": "http://www.ivoa.net/documents/rer/ivo/",
    "voprov": "http://www.ivoa.net/documents/dm/provdm/voprov/",
    "ex": "http://www.example.com/provenance/"
  },
  "activity": {
```

```
    "ex:Process1": {
      "prov:startTime": "2017-04-18T17:28:00",
      "prov:endTime": "2017-04-19T17:29:00",
      "voprov:name": "Process 1"
    }
  },
  "wasGeneratedBy": {
    "_:id4": {
      "prov:time": "2017-05-05T00:00:00",
      "prov:entity": "ivo://example#Public_NGC6946",
      "prov:activity": "ex:Process1"
    }
  },
  "used": {
    "_:id1": {
      "prov:entity": "ivo://CDS/P/DSS2/POSSII#POSSII.J-DSS2.143",
      "prov:activity": "hips:AlaRGB1"
    }
  }
  "entity": {
    "ivo://example#DSS2.143": {
      "voprov:name": "Unprocessed image of NGC6946"
    },
    "ivo://example#Public_NGC6946": {
      "voprov:name": "Processed image of NGC 6946"
    }
  }
}
```

To emphasize the compatibility to the IVOA framework, where the XML VOTable format is a reference to circulate metadata, we define a PROV-VOTABLE master document where all classes' declarations and relations described in PROVN are translated as separated tables.

These VOTable serialisations can be produced using the VOPROV Python module [3] python module, available to the community. See also Section 6.2 and the IVOA Prov-DM Implementation Note (Riebe and Servillat et al., 2017).

This is the VOTable serialization:

```
<?xml version="1.0" encoding="UTF-8"?>
<VOTABLE version="1.2" xmlns="http://www.ivoa.net/xml/VOTable/v1.2" xmlns:ex="http://www.example.com/provenance"
  <RESOURCE type="provenance">
    <DESCRIPTION>Provenance VOTable</DESCRIPTION>
    <TABLE name="Usage" utype="voprov:used">
      <FIELD arraysize="*" datatype="char" name="activity" ucd="meta.id" utype="voprov:Usage.activity"/>
      <FIELD arraysize="*" datatype="char" name="entity" ucd="meta.id" utype="voprov:Usage.entity"/>
      <DATA>
        <TABLEDATA>
          <TR>
            <TD>ex:Process1</TD>
            <TD>ivo://example#DSS2.143</TD>
          </TR>
        </TABLEDATA>
      </DATA>
    </TABLE>
```

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

```
    <TABLE name="Generation" utype="voprov:wasGeneratedBy">
      <FIELD arraysize="*" datatype="char" name="entity" ucd="meta.id" utype="voprov:Generation.entity"/>
      <FIELD arraysize="*" datatype="char" name="activity" ucd="meta.id" utype="voprov:Generation.activity"/>
      <DATA>
        <TABLEDATA>
          <TR>
            <TD>ivo://example#Public_NGC6946</TD>
            <TD>ex:Process1</TD>
          </TR>
        </TABLEDATA>
      </DATA>
    </TABLE>
    <TABLE name="Activity" utype="voprov:Activity">
      <FIELD arraysize="*" datatype="char" name="id" ucd="meta.id" utype="voprov:Activity.id"/>
      <FIELD arraysize="*" datatype="char" name="name" ucd="meta.title" utype="voprov:Activity.name"/>
      <FIELD arraysize="*" datatype="char" name="start" ucd="" utype="voprov:Activity.startTime"/>
      <FIELD arraysize="*" datatype="char" name="stop" ucd="" utype="voprov:Activity.endTime"/>
      <DATA>
        <TABLEDATA>
          <TR>
            <TD>ex:Process1</TD>
            <TD>Process 1</TD>
            <TD>2017-04-18 17:28:00</TD>
            <TD>2017-04-19 17:29:00</TD>
          </TR>
        </TABLEDATA>
      </DATA>
    </TABLE>
    <TABLE name="Entity" utype="voprov:Entity">
      <FIELD arraysize="*" datatype="char" name="id" ucd="meta.id" utype="voprov:Entity.id"/>
      <FIELD arraysize="*" datatype="char" name="name" ucd="meta.title" utype="voprov:Entity.name"/>
      <DATA>
        <TABLEDATA>
          <TR>
            <TD>ivo://example#DSS2.143</TD>
            <TD>Unprocessed image of NGC6946</TD>
          </TR>
          <TR>
            <TD>ivo://example#Public_NGC6946</TD>
            <TD>Processed image of NGC 6946</TD>
          </TR>
        </TABLEDATA>
      </DATA>
    </TABLE>
    <INFO name="QUERY_STATUS" value="OK"/>
  </RESOURCE>
</VOTABLE>
```

This VOTABLE serialisation can be considered as a flat view on the various tables stored in a database implementing the datamodel structure explained in section 2 More examples of serialisation documents are provided in Appendix A **??**.

Such serializations can be retrieved through access protocols (see 5.1) or directly integrated in dataset headers or "associated metadata" in order to provide provenance metadata for these datasets. E.g. for FITS files a provenance extension called "PROVENANCE" could be added which contains provenance information of the workflow that generated the FITS file in one of the serialization formats.

TODO:
SVOM strategy to incorporate provenance as an extension in FITS ? still valid ?

TODO:
Check that this keyword is not already taken.

4.2 Serialization of description classes

The ProvenanceDM includes description classes that can exist before any provenance information is recorded. First, the ActivityDescription class gives information on the activity (name, description, doculink...) and the parameters expected as an input. In addition, UsedDescription and WasGeneratedByDescription classes indicate the expected roles of the input and output entities respectively. Finally, The activity may expect specific kinds of entities as inputs or outputs, for which there may be detailed descriptions stored as EntityDescription records.

The serialization of an ActivityDescription, that includes all those description classes, is based on the IVOA DataLink Service Descriptors for service resources (Dowler and Bonnarel et al., 2015), and can thus be stored as a VOTable (Ochsenbein and Williams et al., 2013). Indeed, a service descriptor points to a service that probably executes an activity using the given input parameters, some of which probably point to entities. One can thus easily translate an ActivityDescription VOTable to a DataLink service descriptor VOTable block, and vice-versa.

The VOTable contains one resource with attributes type="meta" and utype="voprov:ActivityDescription". This resource contains PARAM elements to describe the activity and GROUP elements with additional PARAM elements to describe the input parameters (group name="InputParams"), the input entities (group name="Used") and the output entities (group name="Generated").

The standard PARAM elements for an activity resource correspond to the attributes of the ActivityDescription class (see Section 2.2.4) and may include an Agent name and email. For the input parameters, each ParameterDescription element is mapped to a PARAM element. The mapping is direct as ParameterDescription is based on PARAM. For the input and output entity groups, each related entity is described with a PARAM block where the name is the role of the entity in the scope of the activity, and the expected value is the entity identifier (utype="voprov:Entity.id"). It is possible to reference an input parameter using the ref attribute of PARAM, if an input entity is given as an input parameter to the activity (e.g. the name of a file). The xtype attribute of PARAM can be used to provide the content type (MIME type) of the entity.

Here is an example of an ActivityDescription VOTable that describes an activity to create an RGB image from three red, green, blue images:

```
<VOTABLE xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://www.ivoa.net/xml/VOTable/v1.3" version="1.3"
    xsi:schemaLocation="http://www.ivoa.net/xml/VOTable/v1.3
    http://www.ivoa.net/xml/VOTable/v1.3">
  <RESOURCE ID="make_RGB_image" name="make_RGB_image"
      type="meta" utype="voprov:ActivityDescription">
    <DESCRIPTION>Create an RGB image from 3 images</DESCRIPTION>
    <LINK content-role="doc" href="..."/>
    <PARAM name="label" datatype="char" arraysize="*"
        value="make_RGB_image" utype="voprov:ActivityDescription.label"/>
    <PARAM name="type" datatype="char" arraysize="*"
        value="None" utype="voprov:ActivityDescription.type"/>
    <PARAM name="subtype" datatype="char" arraysize="*"
        value="None" utype="voprov:ActivityDescription.subtype"/>
    <PARAM name="version" datatype="float"
        value="None" utype="voprov:ActivityDescription.version"/>
    <PARAM name="contact_name" datatype="char" arraysize="*"
        value="..." utype="voprov:Agent.name"/>
    <PARAM name="contact_email" datatype="char" arraysize="*"
        value="...@..." utype="voprov:Agent.email"/>
    <GROUP name="InputParams" utype="voprov:Parameter">
      <PARAM ID="RGB" arraysize="*" datatype="char" name="RGB"
          type="no_query" value="RGB.jpg">
        <DESCRIPTION>RGB image name</DESCRIPTION>
      </PARAM>
      <PARAM ID="order" arraysize="*" datatype="char" name="order"
          type="no_query" value="RGB">
        <DESCRIPTION>order of the channels</DESCRIPTION>
        <VALUES>
          <OPTION value="RGB"/>
          <OPTION value="RBG"/>
          <OPTION value="GBR"/>
          <OPTION value="GRB"/>
          <OPTION value="BRG"/>
          <OPTION value="BGR"/>
        </VALUES>
      </PARAM>
    </GROUP>
    <GROUP name="Used" utype="voprov:Used">
      <PARAM arraysize="*" datatype="char" name="R"
          value="R.jpg" utype="voprov:Entity.id" xtype="image/jpeg">
        <DESCRIPTION>Image for red channel</DESCRIPTION>
      </PARAM>
      <PARAM arraysize="*" datatype="char" name="G"
          value="G.jpg" utype="voprov:Entity.id" xtype="image/jpeg">
        <DESCRIPTION>Image for green channel</DESCRIPTION>
      </PARAM>
      <PARAM arraysize="*" datatype="char" name="B"
          value="B.jpg" utype="voprov:Entity.id" xtype="image/jpeg">
        <DESCRIPTION>Image for blue channel</DESCRIPTION>
      </PARAM>
    </GROUP>
    <GROUP name="Generated" utype="voprov:WasGeneratedBy">
      <PARAM arraysize="*" datatype="char" name="RGB" ref="RGB"
          value="RGB.jpg" utype="voprov:Entity.id"  xtype="image/jpeg">
        <DESCRIPTION>RGB image name</DESCRIPTION>
      </PARAM>
    </GROUP>
  </RESOURCE>
```

```
</VOTABLE>
```

5 Accessing provenance information

5.1 Access protocols

We envision two possible access protocols:

- ProvDAL: retrieve provenance information based on given ID of a data entity or activity.
- ProvTAP: allows detailed queries for provenance information, discovery of datasets based on e.g. code version.

5.2 ProvDAL

ProvDAL is a simple data access layer interface (see DALI specification of the VO, Dowler and Demleitner et al., 2013) 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 14.

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.

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

Table 14: ProvDAL request parameters. Options that are **required** to be implemented by ProvDAL services are marked with bold face. Default values are underlined. The parameter names are case-insensitive, but the parameter values are not.

TODO:
KR: I suggest to add here:
Services may restrict the returned data by redirecting `DEPTH=ALL` to e.g. `DEPTH={maxdepth}`, where `{maxdepth}` is an integer defining the maximum depth number that the server allows.

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 11). (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.)

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.

TODO:
What about adding the MODEL-parameter (values: IVOA/W3C)?

Figure 11 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.

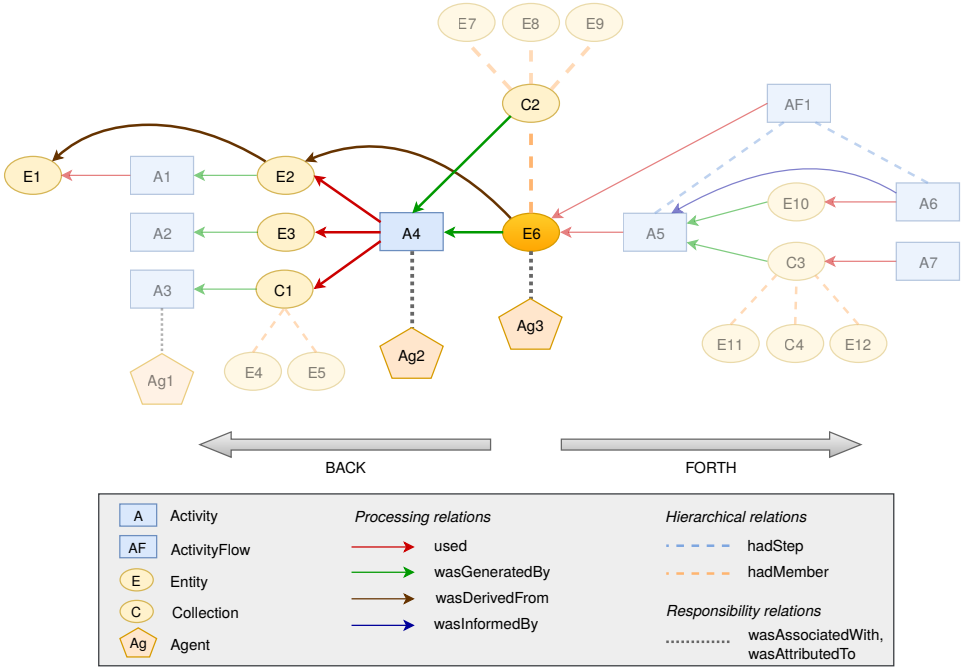

Figure 11: An example provenance graph, highlighting the objects and relations returned from a ProvDAL service with `ID=E6` and `DEPTH=2`. The BACK and FORTH values for DIRECTION are only important for the processing relations (solid lines). Hierarchial (dashed) and responsibility (dotted) relations 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 (Dowler and Demleitner et al., 2013), 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).

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 (Dowler and Demleitner et al., 2013), the parameter names are case-insensitive, but the parameter values are not. E.g. `direcion=FORTH` is allowed, but `DIRECTION=forth` may not work.

5.2.1 ProvDAL example use cases

TODO:
This is a new section.

We provide here a few example use cases for ProvDAL in order to show its usefulness in exploring the provenance of astronomical datasets, processes or the people and projects involved in producing/performing them.

- The RAVE DR4 release contains a main table with stellar properties for each observation of a star. Given the RAVE observation ID, retrieve 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 important information can be filtered out by a client application (e.g. use voprov Python package). If a W3C tool shall be used, one needs to transform the response into a

W3C compliant serialisation (e.g. for loading the result to ProvStore[4] for further processing).

- 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 not any "backwards" information about progenitors, then one needs to track the collection further, i.e. repeat the request for the collection entity.

- 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
  ```

  By using `DIRECTION=FORTH` we can track the dataset and where it was used forward.

- Find all people that were involved in processing a dataset along with their contact data (if available), so that one can ask them for further information.

  ```
  {provdal-base-url}?ID=ex:e1&DEPTH=ALL
  ```

  The ProvDAL request is basically the same as in the first example. From the results the agents need to be filtered. Since the response contains the nodes and relations including all their properties, the contact details for the agents are included as well (if they are stored with the service).

- Retrieve a VOTABLE serialisation of the provenance for an image from a data collection.

  ```
  {provdal-base-url}?ID=myproject:img1&DEPTH=ALL\
  &RESPONSEFORMAT=PROV-VOTABLE
  ```

  We use the RESPONSEFORMAT keyword here to retrieve a VOTABLE.

ProvDAL is meant to be used to retrieve parts of a provenance graph from a provenance web service. It cannot be used to retrieve 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 (see next section).

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

5.3 ProvTAP

//currently updated ... ProvTAP is a TAP service implementing the ProvenanceDM data model. The data model mapping is included in the TAP schema. The mapping of ProvenanceDM classes and attributes onto tables and columns of the schema with the appropriate relationships, datatypes, units, utypes and ucds is done similarly to the PROV-VOTABLE serialization. The query response will result in a single table according to the query. This single table is joining information coming from one or several "provenance" tables available in the database.

A special case is considered where ProvenanceDM and ObsCore are both implemented in the same TAP service and queried together. The TAP response is then providing an Obscore table with a ProvenanceDM extension. We can imagine that in the future this could be hard-coded and registered as an ObsProvTAP service.

TODO:
We need more details here! Output of TAP service is NOT a PROV-VOTABLE by default!

5.4 VOSI availability and capabilities

TODO:
Still needs to be discussed!

According to the DALI specification for VO services (Dowler and Demleitner et al., 2013), a provenance service implementing ProvDAL and/or ProvTAP must provide a VOSI availability interface as well as a capabilities interface with entries for ProvDAL and/or ProvTAP. The `standardIds` for these provenance interfaces are:

```
ivo://ivoa.net/std/ProvenanceDM#ProvDAL
ivo://ivoa.net/std/ProvenanceDM#ProvTAP
```

The capability for a TAP service to support the Provenance DM is expressed by the dataModel element as :

```
<dataModel ivoid="ivo://ivoa.net/std/ProvenanceDM#core-1.0">ProvenanceDM-1.0</dataMo
```

For ProvTAP, the VOSI tables interface also needs to be provided.

6 Use cases – applying the data model

This section presents some general guidelines for applying the data model and specific use cases for which the provenance data model helps to solve certain tasks. Details on specific implementions of the provenance data model

are provided in a separate document, the ProvenanceDM Implementation Note (Riebe and Servillat et al., 2017).

6.1 How to use the data model

TODO:
KR: I think this is a good place for this section. Do we still want to have this section or is everything covered with the new section on entitydescription-serialisation?

- identify entities in your project, i.e. the things you deal with
- identify activities (processes) in your project
- identify reponsible agents (persons and organizations)
- find the relations between them
- possibly 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, dataProduct_type, what kind of entity it is going to be (category)
- ...

6.2 voprov Python package

The voprov Python package is derived from the voprov-package. It can load data and serialize it. The voprov-package supports following features: ...

TODO:
Michele: Please add more explanations!

Example code and serializations are given in the Implementation Note (Riebe and Servillat et al., 2017).

6.2.1 Graphic formats

The voprov python module can also provide provenance information in graphic formats: PNG, SVG and PDF. In the above example, you have to add the following instructions in your python program:

```
dot = prov_to_dot(provdoc, use_labels=True)
dot.write_png('ex1.png')
dot.write_svg('ex1.svg')
dot.write_pdf('ex1.pdf')
```

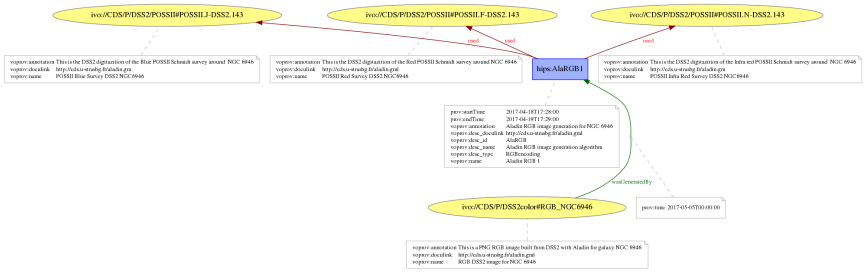

Figure 12: Example: png format@

6.3 Provenance of RAVE database tables (DR4)

The RAVE survey (Radial Velocity Experiment) recorded spectra for about half a million stars. These spectra are processed in a number of steps until the derived properties are published in the RAVE data releases at http://www.rave-survey.org. Providing provenance information for the data, from which spectrum and fibre the data was coming from and which steps were involved in processing the data, can help scientists to understand the data and their restrictions and judge their 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. Provenance information for some major steps of RAVE DR4 was loaded in W3C-compatible PROV-N notation and uploaded to the provenance store at https://provenance.ecs.soton.ac.uk/store/documents/84064/. This allows to view graphs of the workflow by visualising only the main entities, activities and agents with their relations. It shows that the provenance concepts explained in this draft can be applied directly to data obtained from astronomical observations.

We also tested a Django implementation of the classes in this document along with provenance data stored in an SQLite database. This allows to quickly setup a provenance web service which gives the possibility to view all instances of a class or details for a single object, extract provenance information for single entities (backwards in time) and visualise the provenance information. More details about this are available in the implementation notes (Riebe and Servillat et al., 2017).

6.4 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

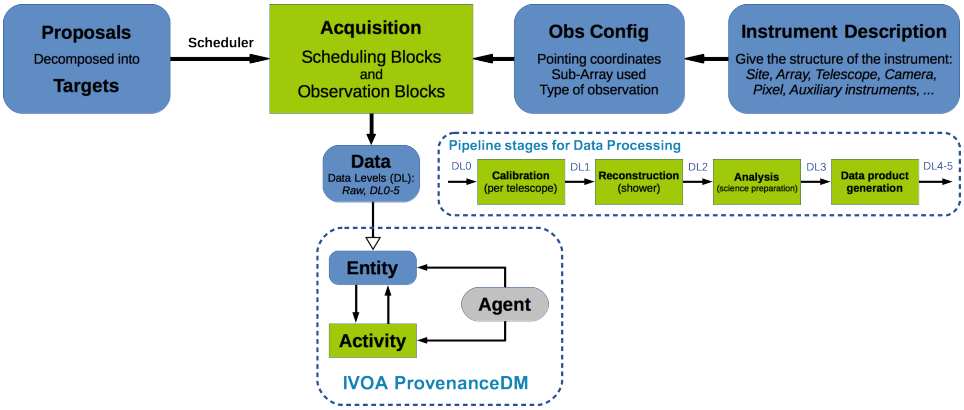

Figure 13: CTA high level data model structure with Pipeline stages and connection to IVOA ProvenanceDM.

products to users that may be unaware of the Cherenkov astronomy specificities. The proposed structure of the metadata is presented in Figure 13.

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 13. 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.
- 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, ...)

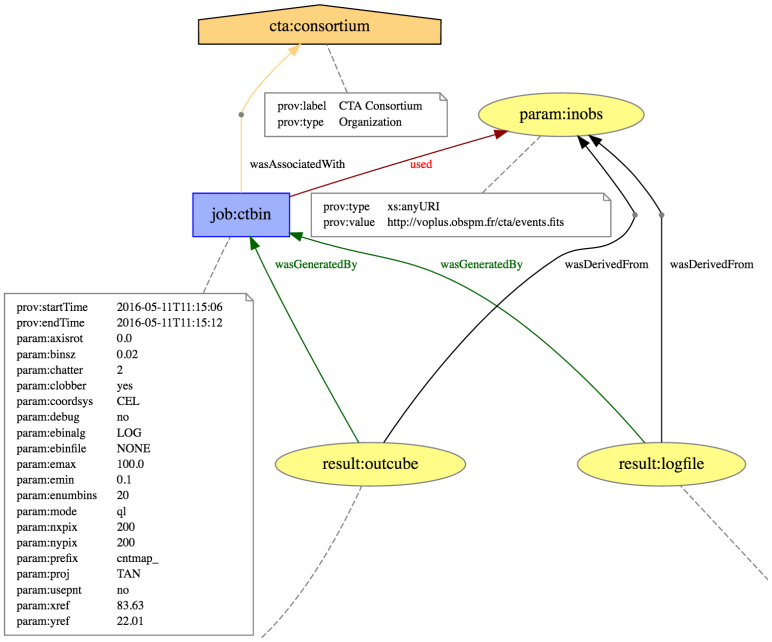

Figure 14: Provenance description of a CTA analysis step.

We tested the tracking of Provenance information using the Python prov package inside OPUS[5] (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 serialisations, as well as a graph visualization (see Figure 14).

6.5 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 codes (CMF_FLUX,SYNSPEC and TURBOSPECTRUM) and atomic linelists from VALD database and specific molecular linelists 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, votables, ...) 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 in another format the description of the provenance of these spectra and on a second hand to select data depending on provenance criteria.

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

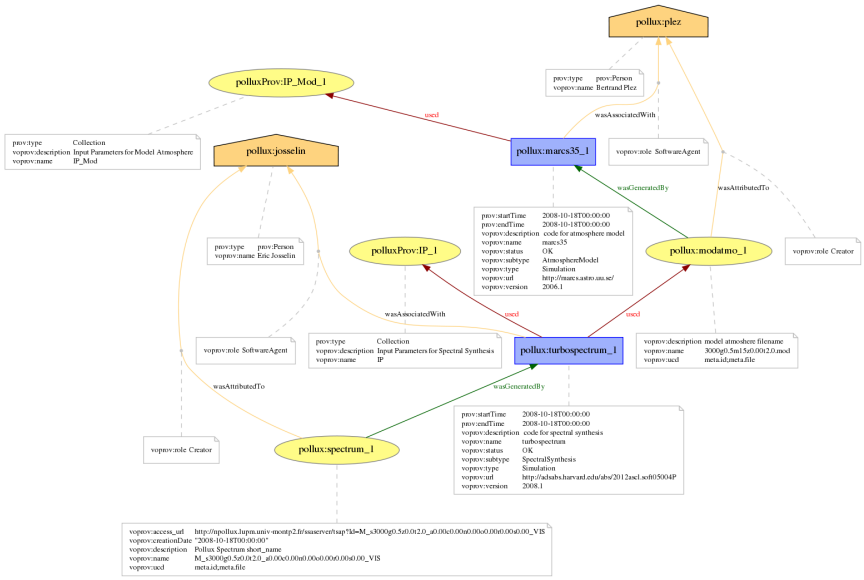

Figure 15: Pollux Example 1

6.6 HiPS use case

HiPS is a new all sky organization of pixel data. It is based on HealPix tesselation of the sky on equal area cells (pixels) for a given HealPix order gathered in tiles. Adaptative resolution is achieved by a hierarchy of tiles at increasing order. Sorting and organization is based on a tree of including directories each of those associated with a tile. HiPS specification has entered the IVOA recommendation process and is becoming an interoperability standard. In the processing chain, HiPS can be seen as a kind of "legacy level" for observational data.

An HiPS dataset can be generated either by Aladin in "hipsgen" mode or by other softwares. The processing distinguishes 3 main different methods for estimating cell values: FIRST or NEAREST neighbour, Mean or Median of the neigbouring pixels. Up to 50 parameters can help to tune the processing, among which can be found the higher resolution HealPix order, sky background value to be substracted, border width or mask to apply to original images to avoid including bad area in the computing, 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 `https://volute.g-vo.org/svn/trunk/projects/dm/provenance/example/HiPS-prov-provn.txt`, the corresponding votable-format is available at `https://volute.g-vo.org/svn/trunk/projects/dm/provenance/example/HiPS-prov-vot.xml`.

Here is an extract of the PROV-N serialization:

```
Entity
( ivo://CDS/P/MAMA/ESO-R,
[
prov:label = "ESO-R MAMA HIPS at CDS",
prov:annotation = "This is the HiPS version of ESO Schmidt survey digitized by Mama
hips:HiPS_properties = "http://cds.u-strasbg.fr/hips/p/mama/eso-r/properties.txt",
voprov:access_reference = "http://CDS/P/MAMA/ESO-R", // as defined in obscore
voprov:doculink = "http://cds.u-strasbg.fr/hips/documentation.html#structure",
voprov:dataproduct_type = "voprov:hips_pixels",
voprov:level = 3
]
)

// Relationship
WasAttributedTo(ivo://CDS/P/MAMA/ESO-R, ivo://cds, prov:role = "voprov:creator")

Agent
(ivo://cds,
[
voprov:Name = "CDS",
voprov:contact = "question@astro.unistra.fr",
prov:type = "Organisation"
]
)
```

6.7 Lightcurves use case

TBD. See Provenance webpage in IVOA Twiki for now.

A Appendix A : Serialisation Examples

Here is a a simple example of serialisation of Provenance DM metadata for describing an Activity of color composition and the entity used as input as well as the resulting RGB image.

The Prov-N format **??** as proposed by the W3C is a text format which allows the description of instances of the main 3 classes, as well as the various relations between each instance involved.

Listing 1: PROV-N serialisation example for a Color composition Activity

```
document
  prefix ivo <http://www.ivoa.net/documents/rer/ivo/>
  prefix hips <http://cds.u-strasbg.fr/data/>
  prefix voprov <http://www.ivoa.net/documents/dm/provdm/voprov/>
```

```
  entity(ivo://CDS/P/DSS2color#RGB_NGC6946, [voprov:annotation="PNG␣RGB␣i
  entity(ivo://CDS/P/DSS2/POSSII#POSSII.J-DSS2.143, [voprov:annotation="␣
  entity(ivo://CDS/P/DSS2/POSSII#POSSII.F-DSS2.143, [voprov:annotation="␣
  entity(ivo://CDS/P/DSS2/POSSII#POSSII.N-DSS2.143, [voprov:annotation="␣
  activity(hips:AlaRGB1, 2017-04-18T17:28:00, 2017-04-19T17:29:00, [vopro
  used(hips:AlaRGB1, ivo://CDS/P/DSS2/POSSII#POSSII.J-DSS2.143, -)
  used(hips:AlaRGB1, ivo://CDS/P/DSS2/POSSII#POSSII.F-DSS2.143, -)
  used(hips:AlaRGB1, ivo://CDS/P/DSS2/POSSII#POSSII.N-DSS2.143, -)
  wasGeneratedBy(ivo://CDS/P/DSS2color#RGB_NGC6946, hips:AlaRGB1, 2017-05
endDocument
```

Here is the transcription of the same metadata in the PROV-Json format. Each class and relation of the provenance model lists its corresponding database tables t-upples grouped by nameof the table.

Listing 2: JSON serialisation example for a Color composition Activity


```
{
  "prefix": {
    "ivo": "http://www.ivoa.net/documents/rer/ivo/",
    "voprov": "http://www.ivoa.net/documents/dm/provdm/voprov/",
    "hips": "http://cds.u-strasbg.fr/data/"
  },
  "activity": {
    "hips:AlaRGB1": {
      "voprov:desc_doculink": "http://cds.u-strasbg.fr/aladin.gml",
      "voprov:desc_id": "AlaRGB",
      "prov:startTime": "2017-04-18T17:28:00",
      "voprov:annotation": "Aladin RGB image generation for NGC 6946",
      "voprov:desc_type": "RGBencoding",
      "voprov:desc_name": "Aladin RGB image generation algorithm",
      "prov:endTime": "2017-04-19T17:29:00",
      "voprov:name": "Aladin RGB 1"
    }
  },
  "wasGeneratedBy": {
    "_:id4": {
      "prov:time": "2017-05-05T00:00:00",
      "prov:entity": "ivo://CDS/P/DSS2color#RGB_NGC6946",
      "prov:activity": "hips:AlaRGB1"
    }
  },
  "used": {
    "_:id1": {
      "prov:entity": "ivo://CDS/P/DSS2/POSSII#POSSII.J-DSS2.143",
      "prov:activity": "hips:AlaRGB1"
    },
    "_:id3": {
      "prov:entity": "ivo://CDS/P/DSS2/POSSII#POSSII.N-DSS2.143",
      "prov:activity": "hips:AlaRGB1"
    },
    "_:id2": {
      "prov:entity": "ivo://CDS/P/DSS2/POSSII#POSSII.F-DSS2.143",
      "prov:activity": "hips:AlaRGB1"
    }
  },
```

```
  "entity": {
    "ivo://CDS/P/DSS2/POSSII#POSSII.J-DSS2.143": {
      "voprov:name": "POSSII Blue Survey DSS2 NGC6946",
      "voprov:annotation": "DSS2 digitization of the Blue POSSII Schmidt
      "voprov:doculink": "http://cds.u-strasbg.fr/aladin.gm"
    },
    "ivo://CDS/P/DSS2/POSSII#POSSII.F-DSS2.143": {
      "voprov:name": "POSSII Red Survey DSS2 NGC6946",
      "voprov:annotation": "DSS2 digitization of the Red POSSII Schmidt s
      "voprov:doculink": "http://cds.u-strasbg.fr/aladin.gml"
    },
    "ivo://CDS/P/DSS2/POSSII#POSSII.N-DSS2.143": {
      "voprov:name": "POSSII Infra Red Survey DSS2 NGC6946",
      "voprov:annotation": "DSS2 digitization of the Infra Red POSSII Sch
      "voprov:doculink": "http://cds.u-strasbg.fr/aladin.gm"
    },
    "ivo://CDS/P/DSS2color#RGB_NGC6946": {
      "voprov:name": "RGB DSS2 image for NGC 6946",
      "voprov:annotation": "PNG RGB image built from DSS2 with Aladin for
      "voprov:doculink": "http://cds.u-strasbg.fr/aladin.gml"
    }
  }
}
```

Here is the mapping obtained for the same data description with the PROV-VOTABLE serialisation format.

Listing 3: PROV-VOTABLE serialisation example for a Color composition Activity

```
<?xml version="1.0" encoding="UTF-8"?>
<VOTABLE version="1.2" xmlns="http://www.ivoa.net/xml/VOTable/v1.2" xmlns
  <RESOURCE type="provenance">
    <DESCRIPTION>Provenance VOTable</DESCRIPTION>
    <TABLE name="Usage" utype="voprov:used">
      <FIELD arraysize="*" datatype="char" name="activity" ucd="meta.id"
      <FIELD arraysize="*" datatype="char" name="entity" ucd="meta.id" ut
      <DATA>
        <TABLEDATA>
          <TR>
            <TD>hips:AlaRGB1</TD>
            <TD>ivo://CDS/P/DSS2/POSSII#POSSII.N-DSS2.143</TD>
          </TR>
        </TABLEDATA>
      </DATA>
    </TABLE>
```

```
<TABLE name="Generation" utype="voprov:wasGeneratedBy">
  <FIELD arraysize="*" datatype="char" name="entity" ucd="meta.id" ut
  <FIELD arraysize="*" datatype="char" name="activity" ucd="meta.id"
  <DATA>
    <TABLEDATA>
      <TR>
        <TD>ivo://CDS/P/DSS2color#RGB_NGC6946</TD>
        <TD>hips:AlaRGB1</TD>
      </TR>
    </TABLEDATA>
  </DATA>
</TABLE>
<TABLE name="Activity" utype="voprov:Activity">
  <FIELD arraysize="*" datatype="char" name="id" ucd="meta.id" utype=
  <FIELD arraysize="*" datatype="char" name="name" ucd="meta.title" u
  <FIELD arraysize="*" datatype="char" name="start" ucd="" utype="vop
  <FIELD arraysize="*" datatype="char" name="stop" ucd="" utype="vopr
  <FIELD arraysize="*" datatype="char" name="annotation" ucd="meta.de
  <FIELD arraysize="*" datatype="char" name="desc_id" ucd="" utype="v
  <FIELD arraysize="*" datatype="char" name="desc_name" ucd="" utype=
  <FIELD arraysize="*" datatype="char" name="desc_type" ucd="meta.cod
  <FIELD arraysize="*" datatype="char" name="desc_doculink" ucd="meta
  <DATA>
    <TABLEDATA>
      <TR>
        <TD>hips:AlaRGB1</TD>
        <TD>Aladin RGB 1</TD>
        <TD>2017-04-18 17:28:00</TD>
        <TD>2017-04-19 17:29:00</TD>
        <TD>Aladin RGB image generation for NGC 6946</TD>
        <TD>AlaRGB</TD>
        <TD>Aladin RGB image generation algorithm</TD>
        <TD>RGB encoding</TD>
        <TD>http://cds.u-strasbg.fr/aladin.gml</TD>
      </TR>
    </TABLEDATA>
  </DATA>
</TABLE>
<TABLE name="Entity" utype="voprov:Entity">
  <FIELD arraysize="*" datatype="char" name="id" ucd="meta.id" utype=
  <FIELD arraysize="*" datatype="char" name="name" ucd="meta.title" u
  <FIELD arraysize="*" datatype="char" name="annotation" ucd="meta.de
  <DATA>
    <TABLEDATA>
```

```
            <TR>
              <TD>ivo://CDS/P/DSS2/POSSII#POSSII.J-DSS2.143</TD>
              <TD>POSSII Blue Survey DSS2 NGC6946</TD>
              <TD> DSS2 digitization of the Blue POSSII Schmidt survey arou
NGC 6946</TD>
            </TR>
            <TR>
              <TD>ivo://CDS/P/DSS2/POSSII#POSSII.F-DSS2.143</TD>
              <TD>POSSII Red Survey DSS2 NGC6946</TD>
              <TD> DSS2 digitization of the Red POSSII Schmidt survey aroun
            </TR>
            <TR>
              <TD>ivo://CDS/P/DSS2/POSSII#POSSII.N-DSS2.143</TD>
              <TD>POSSII Infra Red Survey DSS2 NGC6946</TD>
              <TD> DSS2 digitization of the Infra red POSSII Schmidt survey
NGC 6946</TD>
            </TR>
            <TR>
              <TD>ivo://CDS/P/DSS2color#RGB_NGC6946</TD>
              <TD>RGB DSS2 image for NGC 6946</TD>
              <TD>PNG RGB image built from DSS2 with Aladin for galaxy NGC
            </TR>
          </TABLEDATA>
        </DATA>
      </TABLE>
      <INFO name="QUERY_STATUS" value="OK"/>
    </RESOURCE>
</VOTABLE>
```

B Appendix B : Discussion

B.1 Links, ids

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.

If several copies of a dataset exist, and one of them is corrupted, it would even be useful to know exactly which copy was used by a given activity. This can be modeled already with the existing tools (using a copy-activity), but we doubt that many people would actually need this level of detail.

IVOIDs, DOI's or ORCIDs are potentially good candidates for unique identifiers.

B.2 Description classes

This model was established mainly having a database implementation in mind. However, it may be better in the long run to store provenance with the entities themselves, e.g. as an additional extension in fits-headers.

A model using description classes for defining 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. This *some place* is actually the crucial point here. In an ideal world, "some place" could collect all the descriptions from all the possible datasets and methods in astronomy, but building such a look-up place is a quite challenging task – it will probably never be complete. There's also the issue of persistent identifiers/broken links to consider. Normalisation is useful for closed systems, e.g. for describing the provenance for data produced by a certain pipeline (e.g. MuseWise system) or with workflow tools or when a task needs to be repeated many times. However, the VO is quite the contrary of a closed system and we need to keep an eye on what is actually achievable.

When writing down a simple serialisation of e.g. the provenance for a stacked image using the current model including the description classes, it soon becomes quite cumbersome to define everything twice: first the descriptions, then the instances. This basically doubles the number of entries to describe provenance (unless there is already some place with all the descriptions to which we can refer).

Expressing provenance for a stacked image with this smaller set of classes may be simpler, but on the other hand constructing a database schema becomes much harder. We could leave it to the implementors to choose what is more useful for them. When extracting a serialisation of the provenance information from a provenance service, the attributes of the description classes could be combined with the corresponding activity/entity classes. This will produce some repetition (e.g. many entities may have the same descriptive attributes), but avoid having too many classes and links between them.

B.3 ActivityFlow and viewLevel

Since we introduced *ActivityFlow* mainly for having different view levels, we may want to add an attribute *viewLevel* to the class *ActivityFlow*. However, it is not clear, if viewLevel=0 describes the coarsest or most detailed view. It may happen, when recording provenance information, that first a pipeline activity is defined and only later the detailed steps are described and the

pipeline activity is flagged as an activityFlow with certain steps. Also, when having a detailed description of each activity step, one may later decide to group activities together and define an activityFlow. Therefore, it is not that straightforward to define absolute viewLevel-values. Probably this has to be customized for each project itself.

C Appendix C: Changes from Previous Versions

C.1 Changes from WD-ProvenanceDM-1.0-20161121

- Moved discussion section to appendix, cut code and serialization examples, partially moved to Implementation Note (Riebe and Servillat et al., 2017).
- Added paragraph on VOSI interface
- Added a proposed serialization of description classes
- Added optional attributes *Parameter.min*, *Parameter.max*, *Parameter.option*
- Modified text on the content of EntityDescription, now seen as Entity attributes known before the Entity instance exists.
- Added additional figure for entity-activity relations.
- Added optional attributes *Entity.creationTime* and *EntityDescription.category*
- Rewrite of the ProvDAL section in Section 5.1.
- Moved the figure showing relations between Provenance.Agent and Dataset.Party into Section 3.
- Added serialisation examples to Section 4, voprov-implementation example to Section 6.
- Use voprov:type and voprov:role in Table 11 with example agent roles, i.e. replaced prov:person by Individual and prov:organization by Organization.
- Removed the obscore/dataset attributes from EntityDescription, since they are specific for observations only and are not applicable to configuration entities etc.
- Renamed *label* attribute to *name* everywhere, for more consistency with SimDM naming scheme (*label* is reserved there for SKOS labels).

- Extended the entity role examples in table 6.
- Renamed attribute *Entity.access* to *Entity.rights* for more consistency with DatasetDM etc.
- Moved detailed implementation section from appendix to a separate document (implementation note), shortened the use cases & implementation section.
- More explanations on links to data models in Section 3.
- Introduced subsections for Section 3, added table with SimDM-links.
- Renamed *docuLink* to *doculink*
- Avoid double-meaning of *description* (as reference and free-text description) by renaming the free-text description to *annotation*. No need for an additional description-attribute for reference to the corresponding description class, since it's expressed by the corresponding link in the model anyway.
- Applied similar naming scheme to *Parameter* and *ParameterDescription*-classes
- Renamed Section 6 to stress that it deals with implementations.
- Added links to provn and votable-serialization for HiPS-use case, added first part of provn as example in the HiPS-use case section.
- Corrected attribute names in Table 12.

References

Belhajjame, K., B'Far, R., Cheney, J., Coppens, S., Cresswell, S., Gil, Y., Groth, P., Klyne, G., Lebo, T., McCusker, J., Miles, S., Myers, J., Sahoo, S. and Tilmes, C. (2013), 'PROV-DM: The prov data model', W3C Recommendation.
http://www.w3.org/TR/prov-dm/

Bonnarel, F., Laurino, O., Lemson, G., Louys, M., Rots, A., Tody, D. and the IVOA Data Model Working Group (2015), 'IVOA dataset metadata model', IVOA Working draft.
http://www.ivoa.net/documents/DatasetDM/

Bonnarel, F. and the IVOA Data Model Working Group (2016), 'Provenance data model legacy', Webpage.
http://wiki.ivoa.net/twiki/bin/view/IVOA/ProvenanceDataModelLegacy

Bradner, S. (1997), 'Key words for use in RFCs to indicate requirement levels', RFC 2119.
http://www.ietf.org/rfc/rfc2119.txt

Dowler, P., Bonnarel, F., Michel, L. and Demleitner, M. (2015), 'IVOA datalink', IVOA Recommendation 17 June 2015.
http://www.ivoa.net/documents/DataLink/

Dowler, P., Demleitner, M., Taylor, M. and Tody, D. (2013), 'Data access layer interface, version 1.0', IVOA Recommendation.
http://www.ivoa.net/documents/DALI/20131129/

Hanisch, R., the IVOA Resource Registry Working Group and the NVO Metadata Working Group (2007), 'Resource metadata for the virtual observatory, version 1.12', IVOA Recommendation.
http://www.ivoa.net/documents/latest/RM.html

IVOA Data Model Working Group (2005), 'Data model for observation, version 1.00', IVOA Note.
http://www.ivoa.net/documents/latest/DMObs.html

IVOA Data Model Working Group (2008), 'Data model for astronomical dataset characterisation, version 1.13', IVOA Recommendation.
http://www.ivoa.net/documents/latest/CharacterisationDM.html

Lemson, G., Wozniak, H., Bourges, L., Cervino, M., Gheller, C., Gray, N., LePetit, F., Louys, M., Ooghe, B. and Wagner, R. (2012), 'Simulation Data Model Version 1.0', IVOA Recommendation 03 May 2012, arXiv:1402.4744.
http://adsabs.harvard.edu/abs/2012ivoa.spec.0503L

McDowell, J., Salgado, J., Blanco, C. R., Osuna, P., Tody, D., Solano, E., Mazzarella, J., D'Abrusco, R., Louys, M., Budavari, T., Dolensky, M., Kamp, I., McCusker, K., Protopapas, P., Rots, A., Thompson, R., Valdes, F., Skoda, P., Rino, B., Cant, J., Laurino, O., the IVOA Data Access Layer and Groups, D. M. W. (2016), 'Ivoa spectral data model, version 2.0', IVOA Draft.
http://www.ivoa.net/documents/SpectralDM/

Moreau, L., Clifford, B., Freire, J., Futrelle, J., Gil, Y., Groth, P., Kwasnikowska, N., Miles, S., Missier, P., Myers, J., Plale, B., Simmhan, Y., Stephan, E. and den Bussche, J. V. (2010), 'The open provenance model core specification (v1.1)', Future Generation Computer Systems, 27, (6), 743-756. (doi:10.1016/j.future.2010.07.005), University of Southampton.
http://openprovenance.org/;http://eprints.soton.ac.uk/271449/

Ochsenbein, F., Williams, R., Davenhall, C., Demleitner, M., Durand, D., Fernique, P., Giaretta, D., Hanisch, R., McGlynn, T., Szalay, A., Taylor, M. and Wicenec, A. (2013), 'Votable format definition, version 1.3', IVOA Recommendation.
http://www.ivoa.net/documents/VOTable/

Riebe, K., Servillat, M., Bonnarel, F., Louys, M., Sanguillon, M. and the IVOA Data Model Working Group (2017), 'Provenance implementation note', IVOA Note.
http://volute.g-vo.org/svn/trunk/projects/dm/provenance/implementation-note/