2.2 Extended Model

2.2.1 Class diagram and VO-DML compatibility

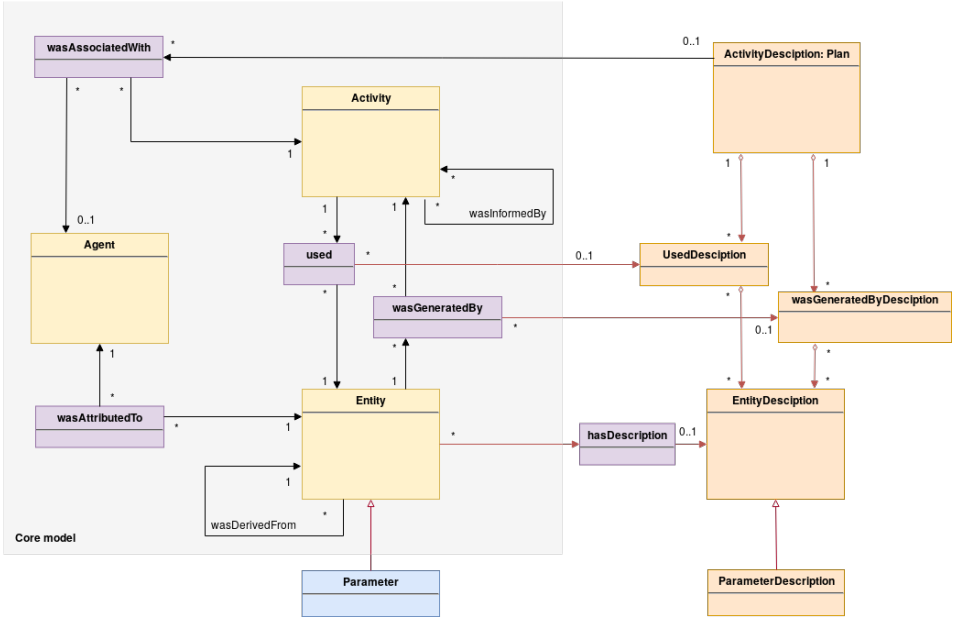

Figure 5: Class diagram showing the main functional features of the Provenance DM. The specialized entities and relations are defined in this section (2.2) and further presented in diagrams 7 and 2.2.3. The various *hasDescription* relations are shown with red arrows.

In the domain of astronomy, certain processes and steps are repeated over and over again, maybe using a different configuration and within a different context. Defining such descriptions allows them to be reused, which is less redundant when performing a series of tasks of the same type. We therefore separate the descriptions of activities from the actual processes and introduce an additional *ActivityDescription* class, which derives from a W3C *prov:Plan*. The relation between the *Activity* and the *ActivityDescription* is *wasAssociatedWith*, which optionally also includes a responsible *Agent*.

For *Entity* we add an *EntityDescription* class. The *EntityDescription* class is linked to the *Entity* by the `description` attribute of that class.

A similar normalization of descriptions of the actual processes and datasets can be found in the IVOA Simulation DM (SimDM, **?**)), which describes simulation metadata. The SimDM classes *Experiment* and *Protocol* correspond to the Provenance terms *Activity* and *ActivityDescription*.

Figure 5 shows the global class diagram. In addition to the core model (Section 2.1), we define specialized entities (Section 2.2.2) and relations (Section 2.2.3) that were identified as useful in one or several use cases. For some

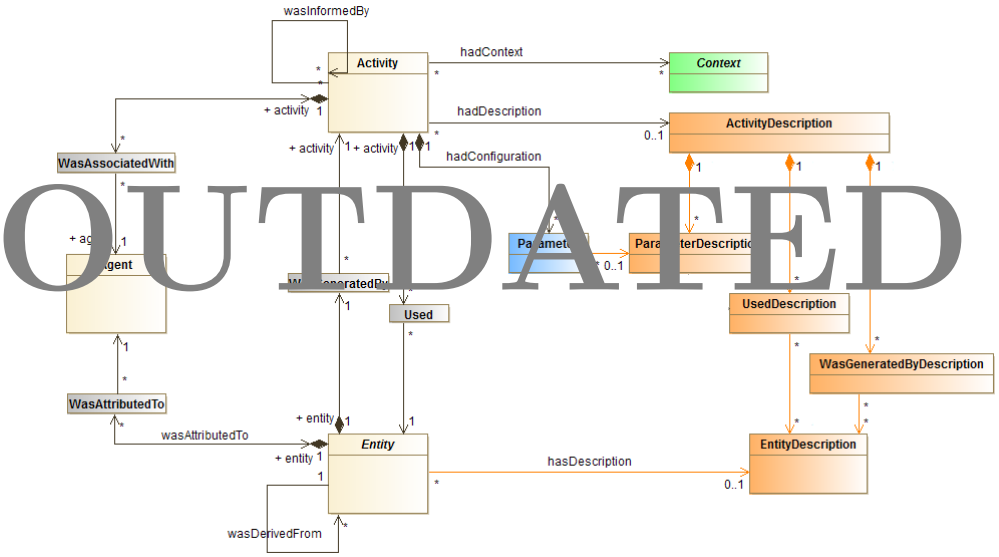

Figure 6: VO-DML compatible version of the class diagram in Figure 5. **Note:** This diagram needs an update to reflect the recent changes.

of the concepts that are well known in the IVOA ecosystem, we propose a detailed structure in order to facilitate the access to such resources and foster interoperability (Sections 2.2.4, 2.2.6 and 2.2.5).

Figure 6 shows a version of the UML diagram applying the VO-DML designing rules and reusing VO-DML IVOA datatypes package. **This is however currently outdated and does not include the changes of the last year.**

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/provenance/vo-dml/ProvenanceDM.html`.

2.2.2 Specialized Entities

The abstraction level of the W3C PROV-DM being high, one of the objective of this IVOA recommendation is to guide the usage of this model in the astronomy context by providing specialized entities that are connected to concepts known in astronomy and relevant to assess the quality and reliability of the exchanged entities.

We first remind the W3C PROV definition of an *Entity*: "An entity is a physical, digital, conceptual, or other kind of thing with some fixed aspects; entities may be real or imaginary."

There are two ways *Entity* objects may be defined: Either they directly carry their value (like numbers, or strings), or they refer to some external object: a file, a table or a table row, or even some physical object like a photo plate. Entities that carry their value form a special subclass, *Parameter* and have the attribute `value` set to their value, while for other entities the pointer to the external object is in the attribute `location`. *Entity* may be linked linked to a *EntityDescription* by the *hasDescription* relation. In case of *Parameter*, the *EntityDescription* will be a *ParameterDescription*.

We already expressed the necessity of adding **Descriptions** to the concepts of Activity and Entity, in order to avoid redundancy and give detail explanations on the method or algorithms that compose the core of an activity. The structuring links of specialized entities is presented in Figure 7. The list given below is not intended to be exhaustive, additional project-dependant entities may be defined when relevant.

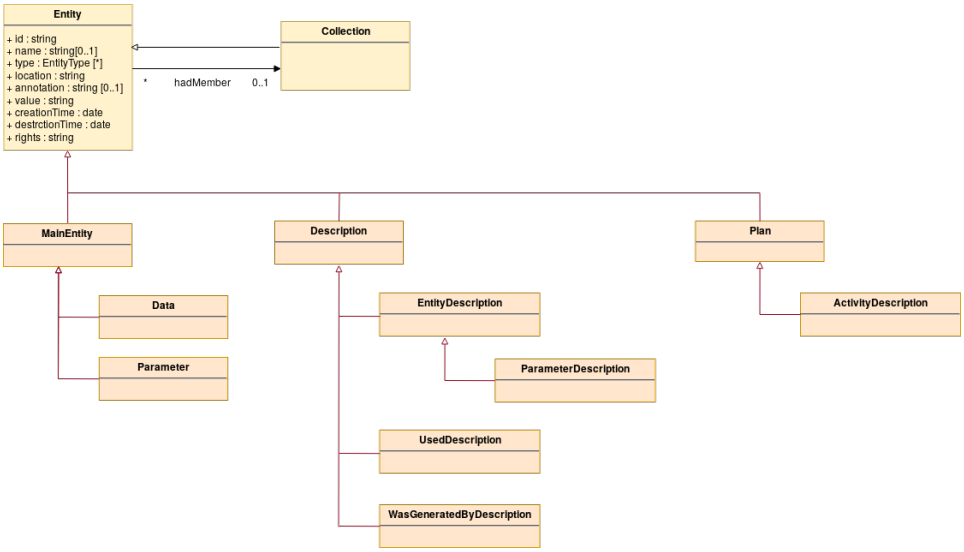

Figure 7: Class diagram showing the structuring links for specialized entities.

It is important to note that specialized entities inherit from *Entity*, all those classes thus share the identifier attribute `id`, as well as the relations of the core model. The `id` attribute must be unique for each different entity so that it can be connected to other concepts using core or specialized relations (see Section 2.2.3). It is possible to attach an Agent to any of those subclasses of *Entity* in order to provide specific contacts for the content of a specialized entity.

MainEntity: Provenance information are expected to be recorded primarily for those main entities, to which we may attach descriptions, detail the

configuration that led to their generation and the context in which they were generated.

Data : digital, machine-readable information in some content that will be used/transformed/analysed. It could be a cell or a column in a table, a file, an image, a cube of data... This *Data Entity* might be bounded to a IVOA *Dataset* record or an IVOA ObsCore record (see also Section B.1 in the Appendix).

Parameter : directly holds its value (typically a number, or a string) directly within the object, e.g. the number of bins desired in the sampling of a signal. This class is further described in Section 2.2.6.

Description: This subclass of entities carries detailed information on the expected behaviour of an activity and on the expected structure and use of an entity.

EntityDescription : describes a category of entities, and contains descriptive information about a normal *Entity* that is known before an entity instance is created (file format, MIME or content type, etc), for example: all files that follow the FITS-LDAC structure and format in a project. This class is further described in Sections 2.2.4 and 2.2.5.

ParameterDescription : is a subclass of *EntityDescription* and contains attributes that describe the value of a *Parameter*. Those attributes are similar to the attributes of the FIELD block in a VOTable (unit, UCD, UType...). This class is further described in Section 2.2.6. It is part of an *ActivityDescription* class.

UsedDescription : describes the roles of expected inputs in an Activity, e.g. a red channel in the creation of an RGB image (see Section 2.2.5).

WasGeneratedByDescription : describes the roles of expected outputs in an Activity, e.g. a master bias in the stacking of a set of bias images (see Section 2.2.5). It is part of an *ActivityDescription* class.

Plan: This W3C subclass of *Entity* carries information on the intended activity to reach its goals. It describes in a structured way the plan prepared for and followed during an activity, and as such, influences directly the activity.

ActivityDescription : explanations on the activity, such as the method used, the algorithm, the source code. This class is further described in Section 2.2.5.

2.2.3 Specialized Relations

In order to distinguish the different usages of entities, we define the corresponding specialized *usage* relations between *Activity* and *Entity*: *hadConfiguration* and *hadContext*. The *ActivityDescription* is connected to the *Activity* through a *wasAssociatedWith* relation, that optionally also links to an *Agent*. In addition, we introduce the *hasDescription* relation that derivates from the W3C relation *wasInfluencedBy*. It connects an *Entity* class to a *Description* class (hence between two *Entity* classes). Those relations are illustrated in Figure 8.

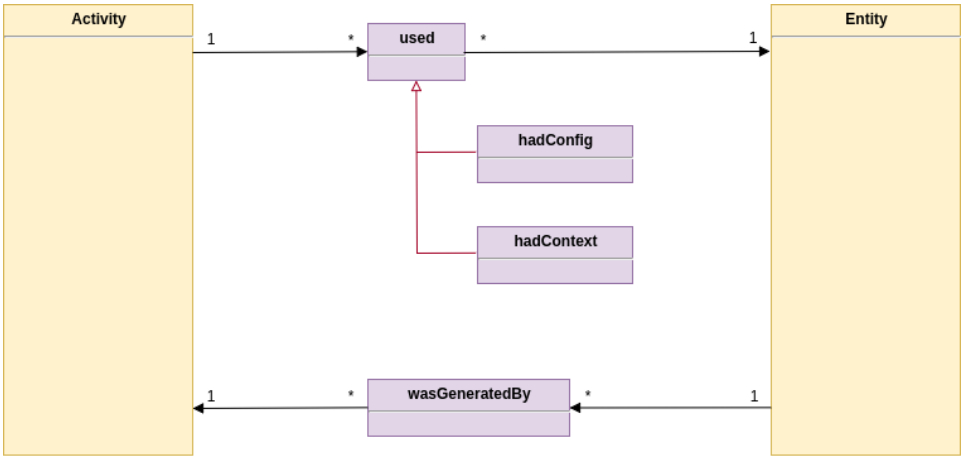

Figure 8: Class diagram showing the usage and generation relations between *Activity* and *Entity*

The `role` attribute in the *used* and *wasGeneratedBy* relations points to the *UsedDescription* and WasGeneratedByDescription classes that further describe the usage or the generation.

2.2.4 EntityDescription

The category of entities can be predefined using a description class *EntityDescription*. This class is meant to store descriptive information about an entity that is known before an *Entity* instance is created. For example, a `format` (e.g. JPG images, FITS, FITS-LDAC, ...) can be common to several entities. However, the size of this image or file cannot be known before it is created. In this example, `format` would be an *EntityDescription* attribute, while size would be a property attached to the *Entity* instance. The *EntityDescription* general attributes are summarized in Table 12. Additional attributes that describe the content of the data could be derived from the Dataset Metadata Model (see Section B.1)

The *EntityDescription* class should *not* contain information about the

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
doculink	url	link to more documentation
Optional attributes:		
content_type	string	MIME type for the content of the entity
format	string	type of container for the entity

Table 12: Attributes of the *EntityDescription* class. Attributes in **bold** must not be null.

usage of the data, in particular, it tells nothing about them being used as input or as output. This kind of information should be provided by the relations (and their relation descriptions) between activities and entities (see Section 2.1.5).

2.2.5 Activity Description classes

The inner working of an activity can be explained by a corresponding *ActivityDescription* class. This could be, for instance, the name of the `code` and its `version` used to perform an activity or a more general description of the underlying algorithm or process. An activity is then a concrete case (instance) that follows the described inner working, with a `startTime` and an `endTime`, and it refers to a corresponding description for further information.

A close concept in the W3C PROV-DM is the *Plan* (defined as a subclass of *Entity*). A plan in PROV-DM is primarily attached to the *Agent* class with *hadPlan* through the *wasAssociatedWith* relation. It is accepted to omit the agent, but it is always supposed that an agent exist. *ActivityDescription* is derived from *Plan*.

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 referred to by the pipeline activity.

Descriptions of the Used and WasGeneratedBy relations. In order to describe more largely an activity, it is common to define the expected inputs

ActivityDescription

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

Table 13: Attributes of the *ActivityDescription* class. Attributes in **bold** must not be null.

and outputs of this activity, i.e. what we expect to store in the Used and WasGeneratedBy relations.

In the case of workflow description models, such as ProvONE (but also Kepler or Taverna for example), input and output **ports** are defined, and can be connected to build a workflow of activities. In ProvONE (**?**), an *ActivityDescription* is restricted to a *Program*, and an *Activity* is an *Execution* associated to a *Program*, with further entities and relations dedicated to workflow descriptions (see SectionB). However, the description of workflows is out of the scope of this document, and the more general concepts we introduce here are the UsedDescription and the WasGeneratedByDescription classes. Those classes are meant to store descriptive information about the usage or generation of an entity that is known before the activity is executed.

In particular, if the `role` attribute is given in those description classes, the corresponding *Used* and *WasGeneratedBy* relations must contain the same `role` value.

A `multiplicity` attribute can indicate that more than one entity may have the same role, e.g. in the case of the stacking of several images, an

undefined number of input images is expected and will share the same role.

UsedDescription

Attribute	Data type	Description
id	string	identifier
type	string	usage type like hadConfiguration or hadContext
name	string	a human-readable name (to be displayed by clients)
annotation	string	additional free text description of the usage
→ default	link	link to the default entity to be used
→ **activityDescription**	link	link to *ActivityDescription*
→ entityDescription	link	link to *EntityDescription*

Table 14: Attributes and references of the *UsedDescription* class. References in the data model are indicated with an arrow (→). Attributes in **bold** must not be null.

WasGeneratedByDescription

Attribute	Data type	Description
id	string	identifier
name	string	a human-readable name (to be displayed by clients)
annotation	string	additional free text description of the generation
→ **activityDescription**	link	link to an *ActivityDescription*
→ entityDescription	link	link to *EntityDescription*

Table 15: Attributes and references of the *WasGeneratedByDescription* class. References in the data model are indicated with an arrow (→). Attributes in **bold** must not be null.

EntityDescription in the context of an Activity. When related to the *UsedDescription* or *WasGeneratedByDescription*, the attributes of EntityDescription (see Section 2.2.4) help to describe the category of entities expected as an input or an output in an activity. For example: the input bias files must be in FITS format, or the red, green and blue channel images must be in PNG or JPEG format.

2.2.6 Parameter and ParameterDescription

For the configuration of activities, but also as input data, often a number of individual values is used. These values are represented by the *Parameter* class, which is a specialized *Entity* class (and therefore may have provenance information). For effectivity, the metadata of parameters are separated in a *ParameterDescription* class, which is shown in table 17. This is modelled in the same way as *FIELD* elements in VOTable (**?**).

In the context of web service resources, a list of input parameters is written in the form of an IVOA DataLink Service Descriptor (**?**), a VOTable resource that contains a group of InputParams with PARAM elements. This connection to Service Descriptors is further developped in Section 3.4.

The *ParameterDescription* class should be used to describe this `value` attribute The attributes of *ParameterDescription* are taken from the FIELD and PARAM elements in the VOTable specification (**?**).

For example, in the case of a processing activity that cleans an image with a sigma-clipping method, the input and output images would be the main entities and the value of the number of sigma for sigma-clipping would be carried by a *Parameter* entity set before running the activity. The corresponding *ParameterDescription* defines the type and range of the expected value for this parameter (using the attributes `datatype`, `min`, `max`, `options`...).

3 Serialization of the provenance data model

3.1 Introduction

Serialization files constitute the building blocks of the client/server dialogs. The provenance information as represented in the data model is split in three main concepts that can be searched following many different relations

Parameter

Attribute	Data type	Description
id	string	parameter unique identifier
value	(value dependent)	the value of the parameter, type depends on `ParameterDescription.datatype` and `xtype`; follows same rules as VOTable TABLEDATA and DALI

Table 16: Attributes of the *Parameter* class. Attributes in **bold** must not be null.

ParameterDescription

Attribute	Data type	Description
id	string	parameter unique identifier
name	string	parameter name
annotation	string	additional free text description
datatype	string	datatype as in VOTable 1.2 and above
arraysize	number	number of values of specified `datatype`, if there is more than one
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
xtype	string	extended datatype as in VOTable 1.2 and above. A list of proposed
Optional attributes:		
min	number	minimum value
max	number	maximum value
options	list	list of accepted values

Table 17: Attributes of the *ParameterDescription* class. Attributes in **bold** must not be null.

between the main 3 classes, *Activity*, *Entity* and *Agent*. The selection of the relations to expose when distributing the provenance information depends on the usage and will be described more extensively in the Implementation Note (**?**) and the links therein.

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 or for an entity, corresponding to a monocolor or RGB image, means expose only the structure of the classes and relation tables and feed them with the related tuples in the database. On the client side, the content of a VO-Provenance serialization document can then be explored and represented using graphical interfaces, as inspired by the Provenance Southampton suite or by customized visualisation tools.

Such serializations can be retrieved through IVOA access protocols (see Section 4), or directly integrated in dataset headers or "associated metadata" in order to provide provenance metadata for these datasets.

For FITS files, a provenance extension called "PROVENANCE" could be added which contains provenance information of the activities that generated the FITS file. This information could be stored directly using one of the serialization formats, for example as a unique cell in an ASCII TABLE extension.

3.2 W3C serialization formats: PROV-N, PROV-JSON and PROV-XML

Serialization formats are proposed in the W3C PROV framework for storing and exchanging the provenance metadata: PROV-N, PROV-JSON,PROV-XML and PROV-RDF, that are defined in **?**, **?**, **?**,and **?** respectively. They can be reused here as well for serializations of our data model.

The W3C compatible serializations use the following namespaces:

- `http://www.w3.org/ns/prov#` (suggested prefix `prov`) for classes and attributes available in the W3C model, and
- `http://http://www.ivoa.net/documents/dm/provdm/voprov/` (suggested prefix `voprov`) for attributes specific to the IVOA provenance model.

Mappings to the W3C names to be used are given in Tables 1, 2, 3, 4, 5 and 6.

The specialized entities defined in Section 2.2.2 must be written as Entity instances and the attribute `prov:type` must be set to the category of the specialized entity e.g. `voprov:Data`, `voprov:Parameter`, `voprov:ActivityDescription`. This is also the rule for *Collection*, as done in W3C PROV-DM. We note here that several `prov:type` values can be provided.

The specialized relations defined in Section 2.2.3 must be written as a W3C relation (e.g. used relation between an Activity and and Entity, or the more general wasInfluencedBy relation between all classes). The attribute `prov:type` must be set to the name of the specialized relation, e.g. `voprov:hadDescription` or `voprov:hasConfiguration`.

Additionally, the following rules must be respected:

- attribute `voprov:name` → `prov:label`
- attribute `voprov:annotation` → `prov:description`
- *ActivityDescription*: also add `prov:type = 'prov:Plan'`

Here is an example of a serialization instance document for an entity being processed by an activity, in PROV-N notation:

```
document
  prefix ivo <http://www.ivoa.net/documents/rer/ivo/>
  prefix voprov <http://www.ivoa.net/documents/dm/provdm/voprov/>
  prefix prov <http://www.w3.org/ns/prov#>
  prefix ex <http://www.example.com/provenance/>

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

Here is the same example in PROV-JSON format:

```
{
  "prefix": {
    "ivo": "http://www.ivoa.net/documents/rer/ivo/",
    "voprov": "http://www.ivoa.net/documents/dm/provdm/voprov/",
    "prov": "http://www.w3.org/ns/prov#",
    "ex": "http://www.example.com/provenance/"
  },
  "activity": {
    "ex:Process1": {
      "prov:startTime": "2017-04-18T17:28:00",
      "prov:endTime": "2017-04-19T17:29:00",
      "prov:label": "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://example#DSS2.143",
      "prov:activity": "hips:AlaRGB1"
    }
  },
  "entity": {
    "ivo://example#DSS2.143": {
      "prov:label": "Unprocessed image of NGC6946",
      "prov:type": "voprov:Data"
    },
```

```
    "ivo://example#Public_NGC6946": {
      "prov:label": "Processed image of NGC 6946",
      "prov:type": "voprov:Data"
    }
  }
}
```

3.3 VOTable format for Provenance metadata

To emphasize the compatibility to the IVOA framework, where the XML-based VOTable format is a reference to circulate metadata, we define a VOTable mapping specification. All classes' declarations and relations described for this data model are translated into separated tables, one for each class of the model. All attributes of these classes are translated to columns, i.e. VOTable FIELDS. In addition, the specification defines the VOTable values of the FIELD and PARAM attributes `ucd`, `datatype`, `utype`, `unit`, `description`, etc.

This can be appropriately used for two goals:

- Publishing full provenance metadata for data collections in VOTable format. This can be produced by data processing workflows or as output of databases containing provenance metadata.
- Providing the backbone for the TAP schema describing IVOA provenance metadata which is used for ProvTAP

The VOTable serialization can be considered as a flat view on the various tables stored in a database implementing the data model structure explained in Section 2. More examples of serialization documents are provided in Appendix A. It is possible to create separate tables for each specialized entity, where the TABLE tag must have the name attribute set to the name of this specialized entity, and `utype=voprov:Entity`.

A VOTable serialization can be produced using the `voprov` python module, available to the community, as mentioned in see also in Implementation Note (**?**).

Here is a VOTable document transcription of the serialization example given above in PROV-N and PROV-JSON:

```
<?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"
  xmlns:ivo="http://www.ivoa.net/documents/rer/ivo/"
  xmlns:voprov="http://www.ivoa.net/documents/dm/provdm/voprov/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ivoa.net/xml/VOTable/v1.2
  ↪  http://www.ivoa.net/xml/VOTable/VOTable-1.2.xsd">
```