International
Virtual
Observatory
Alliance

IVOA Provenance Data Model
Version 1.0

IVOA Proposed Recommendation 2019-07-19

Working group
DM
This version
http://www.ivoa.net/documents/ProvenanceDM/20190719
Latest version
http://www.ivoa.net/documents/ProvenanceDM
Previous versions
WD-ProvenanceDM-1.0-20190614.pdf
PR-ProvenanceDM-1.0-20181015.pdf
WD-ProvenanceDM-1.0-20180530.pdf
WD-ProvenanceDM-1.0-20170921.pdf
WD-ProvenanceDM-1.0-20161121.pdf
ProvDM-0.2-20160428.pdf
ProvDM-0.1-20141008.pdf
Author(s)
Mathieu Servillat, Kristin Riebe, Catherine Boisson, François Bonnarel, Anastasia Galkin, Mireille Louys, Markus Nullmeier, Nicolas Renault-Tinacci, Michèle Sanguillon, Ole Streicher
Editor(s)
Mathieu Servillat

Abstract

This document describes how provenance information 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), a document (e.g. an article, a technical note) or a device (e.g. a camera, a telescope), learn about the people and organizations involved in a project and assess the reliability, quality as well as the usefulness of the dataset, document or device for her own scientific work.

Status of this document

This is an IVOA Proposed Recommendation made available for public review. It is appropriate to reference this document only as a recommended standard that is under review and which may be changed before it is accepted as a full Recommendation.

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

Contents

Acknowledgments

The Provenance Working Group acknowledges support from the Astronomy ESFRI and Research Infrastructure Cluster – ASTERICS project[2], funded by the European Commission under the Horizon 2020 Programme (GA 653477). This document has been developed in part with support from the German Astrophysical Virtual Observatory, funded by BMBF Bewilligungsnummer 05A14BAD and 05A08VHA. Additional funding was provided by the INSU (Action Spécifique Observatoire Virtuel, ASOV), the Action Fédératrice CTA at the Observatoire de Paris, the Paris Astronomical Data Centre (PADC), and the E-Info-Astro project (BMBF 05AI7BA2).

Thanks to: Karl Kosak, Johan Bregeon, Julien Lefaucheur and Jose Enrique Ruiz for the binding to the Cerenkov Telescope Array (CTA) project, Gerard Lemson and Laurent Michel for the VO-DML expression of the data model, Markus Demleitner, Harry Enke, Florian Rothmaier, Jochen Klar and Adrian Partl, for fruitful discussions, remarks and comments at different stages during the preparation of this specification.

We thank the participants of the Provenance Week 2018 in London, in particular Michael Johnson for joining one of our meeting.

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.

[2] http://www.asterics2020.eu/

1 Introduction

In this document, we propose an IVOA standard data model (DM) for describing the provenance of astronomical data. How this specification of the Provenance model can be implemented is developed in a companion document to be published as an IVOA Note (Servillat and Riebe et al., 2017).

The provenance of scientific data is a part of the open publishing policy for science data and follows some of the FAIR principles for data sharing (Wilkinson and Dumontier et al., 2016).

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, light curves), any value, logs, documents, as well as physical objects such as instruments, detectors or photographic plates. The activities correspond to processes like an observation, a simulation, processing steps (image stacking, object extraction, etc.), execution of data analysis code, publication, etc. The people involved can be for example individual persons (observer, publisher, etc.), groups or organisations, i.e. any agent related to an activity or an entity.

An example for activities, entities and agents as they can be discovered backwards in time is given in Figure 1.

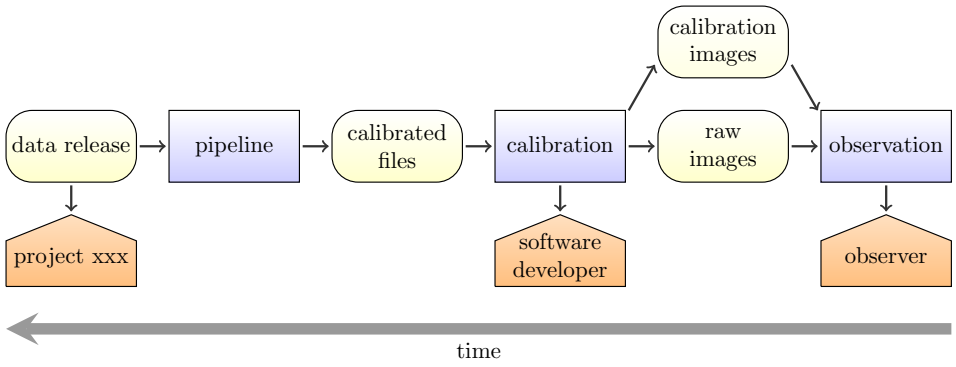

Figure 1: An example graph of provenance discovery. Starting with a released dataset (left), the involved activities (blue boxes), progenitor entities (yellow rounded boxes) and responsible agents (orange pentagons) are discovered.

1.1 Goal of the provenance model

The goal of this Provenance DM is to describe how provenance information arising from astronomy projects can be modelled, stored and exchanged. Its scope is mainly modelling of the flow of data, of the relations between pieces of data, and of processing steps. However, the Provenance DM is sufficiently abstract that its core pattern could be applied to any kind of process related to either observation or simulation data.

Information attached to observation activities such as ambient conditions and instrument characteristics provide useful information to assess the quality and reliability of the generated entities. Contextual information during the execution of processing activities (computer structure, nodes, operating system used, etc.) can also be relevant for the description of the main entities generated. This complementary information should be included in the form of metadata or additional entities connected to an activity. However, the precise structure and modelling of this information is out of the scope of this document.

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 reliability and get enough information to be able to trace back its history as far as required or possible.

Provenance information can be exposed with different granularity. A specific project has to decide this granularity. The granularity and amount of provenance information provided depends on the available information, the needs of the project and the intended usage of this information.

This flexible approach has an impact on the interoperability between different services as this level of detail is not known a priori. The objective of the model is to propose a general structure for the provenance information. In addition, proposed vocabularies of reserved words help to further formalize the detailed provenance information.

The following list is a collection of use cases addressed by the Provenance DM.

A: Traceability of products

Track the lineage of a product back to the raw material (backwards search), show the workflow or the data flow that led to a product.
Examples:

- Having a dataset, find the main progenitors and in particular locate the raw data.

- Find out what processing steps have been already performed for a given dataset: Is an image already calibrated? What about dark field subtraction? Were foreground stars removed?
- Find out if a filter to remove atmospheric background muons has been applied.

B: Acknowledgement and contact information

Find the people involved in the production of a dataset, the people/organizations/institutes that one may want to acknowledge 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 can I contact to get information?
- Find out who was on shift for data taking for a given dataset
- I have a question about column xxx in a data table. Who can I ask about that?

C: Quality and Reliability assessment

Assess the quality and reliability of an observation, production step or dataset, e.g. based on detailed descriptions of the processing steps and manipulated entities.
Examples:

- Get detailed information on the methods/tools/software that were involved: What algorithm was used for Cherenkov photon reconstruction? How was the stacking of images performed?
- Check if the processing steps (including data acquisition) went "well": Were there any warnings during the data processing? Any quality control parameters?
- Extract the ambient conditions during data acquisition (cloud coverage? wind? temperature?)
- Is the dataset produced or published by a person/organisation I can trust?

D: Identification of error location

Find the location of possible error sources in the generation of a product. This is connected to use cases described in section C above, but implies an access to more information on the execution such as configuration or execution environment.
Examples:

- I found something strange in an image. Was there anything strange noted when the image was taken? a warning during the processing?
- Which pipeline version was used, the old one with a known bug for treating bright objects or a newer version?
- What was the execution environment of the pipeline (operating system, coding language version, ...)?
- What was the detailed configuration of the pipeline? were the parameters correctly set for the image cleaning step?

E: Search in structured provenance metadata

Use Provenance criteria to locate datasets (forward search), e.g. finding all images produced by a certain processing step or derived from data which were taken by a given facility.
Examples:

- Find more images that were produced using the same version of the CTA pipeline.
- Get an overview of all images reduced with the same calibration dataset.
- Are there any more images attributed to this observer?
- Find all datasets generated using this given algorithm, with this given configuration, for this given step of the data processing.
- Find all generated data files that used incorrectly generated file X as an input, so that they can be marked for re-processing
- Extract all the provenance information of a SVOM light curve or spectrum to reprocess the raw data with refined parameters.

General Remarks

In addition to those use cases, if the stored information is sufficiently fine grained, it is possible to enable the **reproducibility** of an activity or sequence of activities, with the exact same configuration and exact same conditions.

Another important usage of provenance information is to assess the **pertinence of a product for scientific objectives**, which can be facilitated through the selection of the relevant provenance information attached to an entity that is delivered to a science user.

1.2 Requirements and best practices

1.2.1 Model requirements

This document was developed with these general requirements in mind:

- Provenance information must be formalized following a standard model, with corresponding standard serialization formats.
- Provenance information must be machine readable.
- Provenance data model classes and attributes should make use of existing IVOA standards.
- Provenance information should be serializable into the W3C provenance standard formats (PROV-N, PROV-XML, PROV-JSON) with minimum information loss.
- Entities, Activities and Agents must be uniquely identifiable within a domain.

1.2.2 Best practices

The following additional points are recommended when managing provenance information within the VO context:

- The reliability of provenance information should be ensured (e.g. by an authority endorsing the information, or by provenance of provenance).
- Provenance metadata for a given entity should contain information to find immediate progenitor(s).
- An entity should be linked to the activity that generated it.
- Activities should be linked to input entities.
- Activities should point to output entities.
- Provenance information should make it possible to derive the logical sequence of activities.
- All activities and entities are recommended to have contact information and contain a (short) description or link to a description.

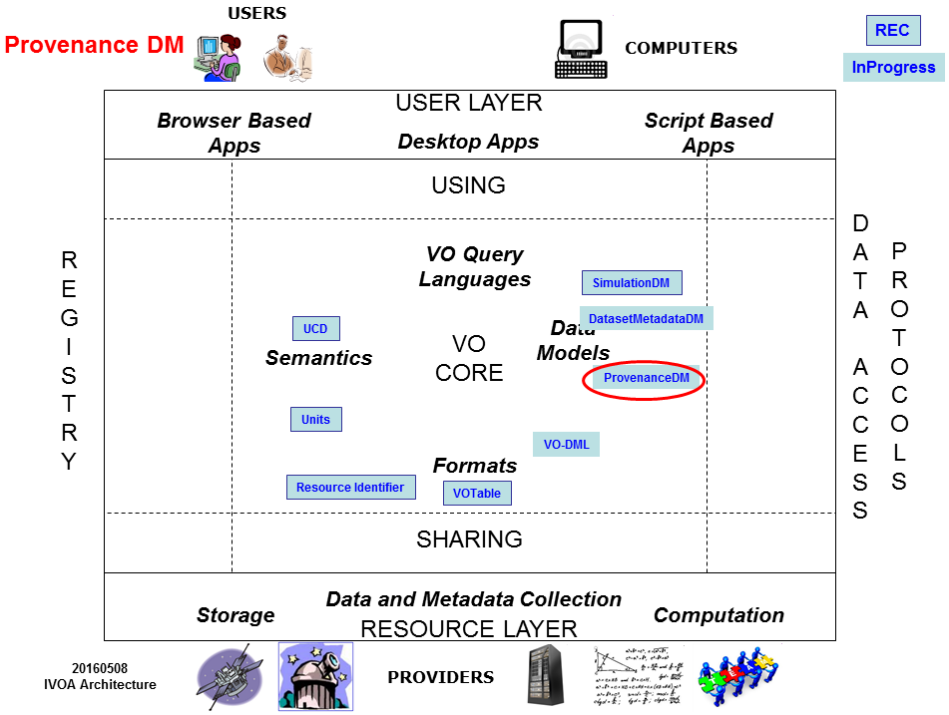

Figure 2: Architecture diagram for the Provenance Data Model. It is based on existing concepts defined in existing IVOA data models, and existing formats and semantics and fully integrated in the IVOA framework

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 for providing astronomical data. Even if it borrows the main general concepts from 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 UCD (semantic tags), VOUnits (standard expression of units for the VO) and service design; and it 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 specifically dedicated to spectral data was

then designed (Spectral Data Model, McDowell and Tody et al., 2007), as well as a fully generic characterisation data model of the measurement axes of data (Characterisation Data Model, Louys and Richards et al., 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 their 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 descriptions. Despite the attempts at an UML description of the model and writing XML serialization examples, the IVOA efforts 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, the 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 (OPM) (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 are 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 PROV Data Model, making it less abstract and more specific, or extending it where necessary.

The W3C model 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 for each entity, agent or relation using the attributes `prov:type` and `prov:role`.

2 The IVOA Provenance data model

2.1 Overview and class diagram

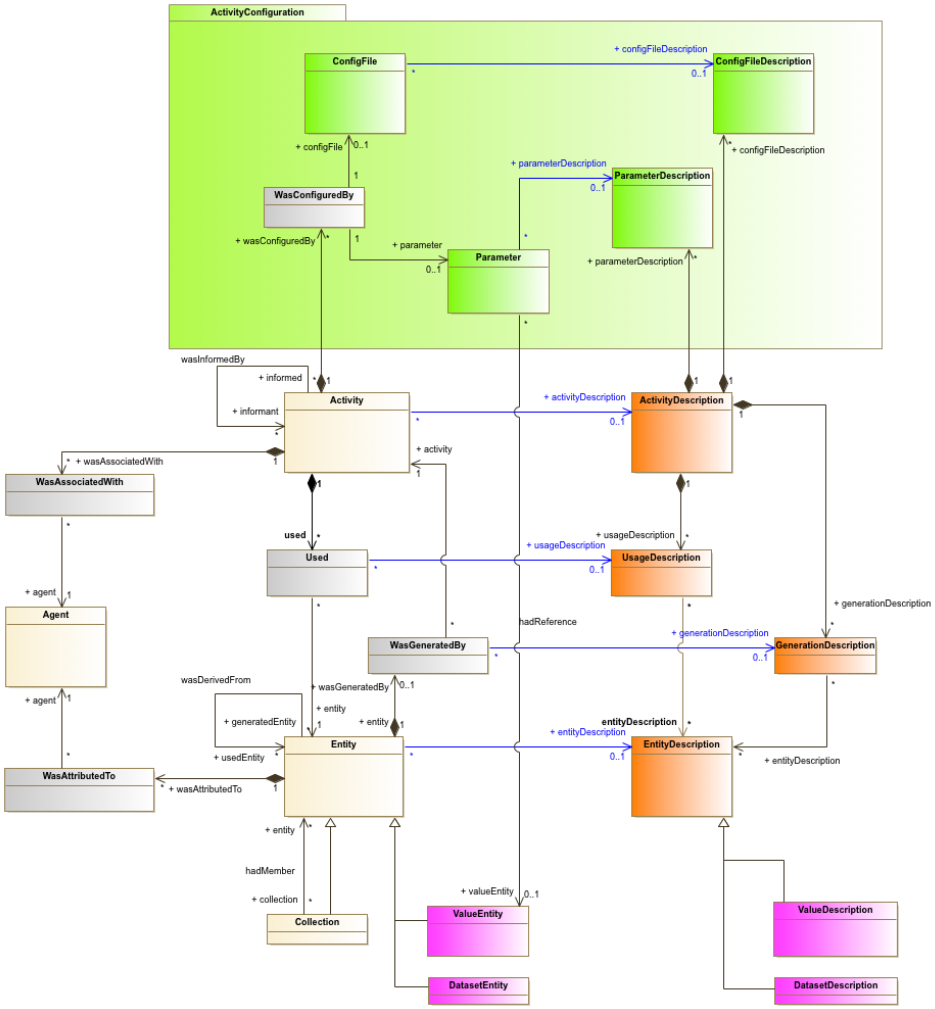

Figure 3: Overview class diagram of the IVOA Provenance Data Model. The core part in yellow is based on W3C PROV definitions where relations are shown in grey. It is extended by a Description part (orange), specific types of entities (red) and an optional Activity Configuration package (green). A full diagram with attributes is shown in Section 3, Figure 8

The IVOA Provenance DM is based on the the PROV-DM recommendation (Belhajjame and B'Far et al., 2013) of the World Wide Web Consortium (W3C), that provides the core elements of the model (see Sections 2.2 to 2.4). In this context, the provenance of something is a sequence of activities using and generating entities run by agents.

The model includes in addition Description classes (see Section 2.5) to provide information common to several elements; Specific types of *Entity* classes commonly used in astronomy (see Section 2.6); and an optional *ActivityConfiguration* package (see Section 2.7).

The IVOA Provenance DM is a class data model that follows the VO-DML designing rules (Lemson and Laurino et al., 2018). It is represented as a UML class diagram: an overview diagram is shown in Figure 3, and a full diagram with attributes is shown in Appendix 3, Figure 8.

2.2 Entity and Activity classes

The core classes and relations of the IVOA Provenance DM are presented in Figure 4. Traceability (see goal A in Section 1.1) is enabled by chaining entities and activities, which are the building blocks of the history graph.

2.2.1 Entity and Collection classes

An **entity** is a physical, digital, conceptual, or other kind of thing with some fixed aspects (W3C PROV-DM §5.1.1).

The *Entity* class in the model have the attributes given in Table 1.

Entities in astronomy are usually astronomical or astrophysical datasets in the form of images, tables, numbers, etc. But they can also be log files, files containing system information, any input or output value, environment variables, ambient conditions, or, in a wider sense, observation proposals, scientific articles, or manuals and other documents. Though the focus is on digital entities in this document, entities can also refer to devices that may be linked to digital entities, such as e.g. tools, instruments, detectors, photographic plates.

A **collection** is an entity that provides a structure to some constituents that must themselves be entities (W3C PROV-DM §5.6.1). These constituents are said to be member of the collections. They are connected in the model with a *hadMember* relation.

2.2.2 Activity class

An **activity** is something that occurs over a period of time and acts upon or with entities; it may include consuming, processing, transforming, modifying, relocating, using, or generating entities (W3C PROV-DM §5.1.2).

The *Activity* class in the model have the attributes given in Table 2.

Activities in astronomy include all steps from obtaining data to the reduction of images and production of new datasets, such as 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.

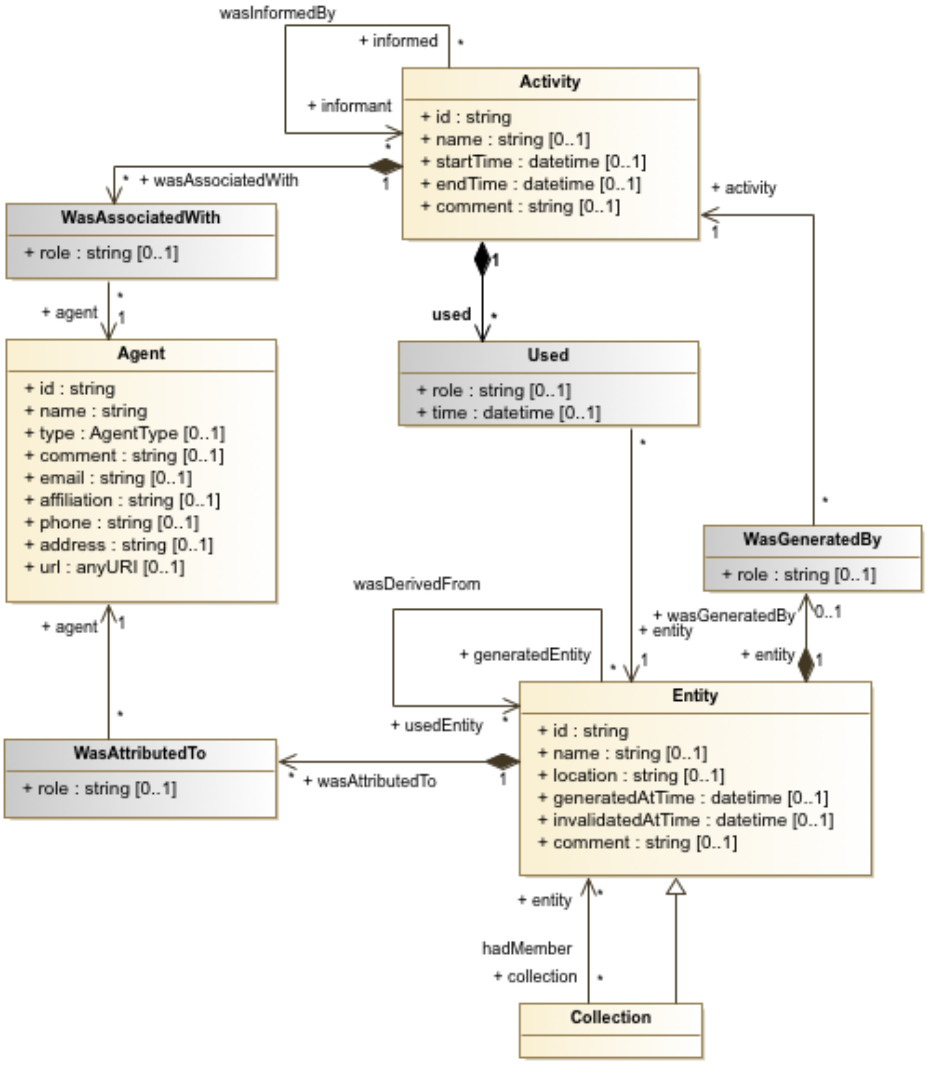

Figure 4: Core classes and relations. Attributes for these classes are detailed in tables found in Sections 2.2 to 2.4.

2.3 Entity-Activity relations

Each entity is usually a result of an activity, expressed by a link from the entity to its generating activity, and can be used as input for (many) other activities. Thus the information on whether data is used as input or was produced as output of some activity is given by the *relations* between activities and entities. Tracking those relations answers one of the main objective of the model (see goal A in Section 1.1).

Entity

Attribute	Data type	Description
id	string	a unique identifier for this entity
name	string	a human-readable name for the entity
location	string	a path or spatial coordinates, e.g. a URL, latitude-longitude coordinates on Earth, the name of a place.
generatedAtTime	datetime	date and time at which the entity was created (e.g. timestamp of a file)
invalidatedAtTime	datetime	date and time of the destruction, cessation, or expiry of the entity. The entity is no longer available for use (or further invalidation) after invalidation.
comment	string	text containing specific comments on the entity

Table 1: Attributes of the *Entity* class. Attributes in **bold** must not be null.

Activity

Attribute	Data type	Description
id	string	a unique id for this activity
name	string	a human-readable name (to be displayed by clients)
startTime	datetime	start of an activity
endTime	datetime	end of an activity
comment	string	text containing specific comments on the activity

Table 2: Attributes of the *Activity* class. Attributes in **bold** must not be null.

2.3.1 Used class

Usage is the beginning of utilizing an entity by an activity. Before usage, the activity had not begun to utilize this entity and could not have been affected by the entity (W3C PROV-DM §5.1.4).

Usage is implemented in the model by a class *Used* that connects *Activity* to *Entity* and contains the attributes in Table 3.

For example, an activity "calibration" used entities with the roles "calibration data" and "raw images".

Used

Attribute	Data type	Description
role	string	function of the entity with respect to the activity
time	datetime	time at which the usage of an entity started

Table 3: Attributes of the *Used* relation class.

The `time` of the usage can be specified, and must be between the `startTime` and the `stopTime` of the corresponding activity.

The *Used* class is closely coupled to the *Activity* by a composition (see B.5). Any given entity can be used by more than one activity.

2.3.2 WasGeneratedBy class

Generation is the completion of production of a new entity by an activity. This entity did not exist before generation and becomes available for usage after this generation (W3C PROV-DM §5.1.3).

Generation is implemented in the model by a class *WasGeneratedBy* that connects *Entity* to *Activity* and contains the attributes in Table 3.

For example, the entity "raw_image.fits" was generated by the activity "observation" with the role "raw image".

WasGeneratedBy

Attribute	Data type	Description
role	string	function of the entity with respect to the activity

Table 4: Attributes of the *WasGeneratedBy* relation class.

As the *Entity* class has an attribute `generatedAtTime`, there is no additional time attribute in this relation.

The *WasGeneratedBy* relation is closely coupled with the *Entity* via a composition (see B.5). An entity can be generated by only one activity, so the multiplicity is 1 or 0 between *Entity* and *WasGeneratedBy*.

2.3.3 Roles in Entity-Activity relations

The `role` of an entity within an activity should be provided. Roles in *Entity-Activity* relations are free text attributes.

The `role` cannot be an attribute of the *Entity* class, since the same entity (e.g. a specific file containing an image) may play different roles with different activities.

In some cases the role is mandatory to distinguish two input entities. For example, an activity for dark-frame 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.

Several entities may play the same role for an activity. For example, many image entities may be used as science-ready-images for an image stacking process.

2.3.4 WasDerivedFrom relation

A **derivation** is a transformation of an entity into another, an update of an entity resulting in a new one, or the construction of a new entity based on a pre-existing entity (W3C PROV-DM §5.2.1).

Derivation is an optional relation *wasDerivedFrom* in the model, that connects an instance of *Entity* to another instance.

For example, the entity "calibrated_image.fits" was derived from the entity "raw_image.fits"

This relation makes it possible to visualize independently the flow of entities, e.g. a dataflow. It does not need a priori a specific class or table in an implementation, but it provides a way to expose partial information that follow the general chain *WasGeneratedBy-Activity-Used* where the activity may be an empty instance because it is unknown or irrelevant.

2.3.5 WasInformedBy relation

Communication is the exchange of information (some unspecified entity) by two activities, one activity using some entity generated by the other (W3C PROV-DM §5.1.5).

Communication is an optional relation *wasInformedBy* in the model, that connects an instance of *Activity* to another instance.

For example, the activity "calibration" wasInformedBy the activity "pipeline".

This relation makes it possible to visualize independently the flow of activities as they occurred, which may be the result of the execution of a workflow. It does not need a priori a specific class or table in an implementation, but it provides a way to expose partial information that follow the general chain *Used-Entity-WasGeneratedBy* where the entity may be an empty instance because it is unknown or irrelevant.

2.4 Agent and relations to Agent

A contact information is needed in case more information about a certain activity or entity is required, but also in order to know who was involved and to fulfil the Acknowledgement objective (see goal B in Section 1.1).

2.4.1 Agent class

An **agent** is something that bears some form of responsibility for an activity taking place or for the existence of an entity (W3C PROV-DM §5.3.1).

The *Agent* class in the model has the attributes given in Table 5.

An Agent is generally someone 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, a project or an institute.

It is recommended to use organizational agents and agents with generic contacts.

Agent

Attribute	Data type	Description
id	string	unique identifier for an agent
name	string	a common name for this agent; e.g. first name and last name; project name, pipeline team, data center.
type	AgentType	type of the agent as given in Table 6
comment	string	text containing specific comments on the agent
email	string	contact email of the agent
affiliation	string	affiliation of the agent
phone	string	phone number
address	string	address of the agent
url	anyURI	reference URL to the agent

Table 5: Attributes of the *Agent* class. Attributes in **bold** must not be null.

For each agent a `name` must be specified. Other attributes can help locate or contact the agent (`email`, `affiliation`, `phone`, `address`). Not every project will need them; e.g. an advanced system may use permanent identifiers (e.g. ORCIDs, identities in federations, etc) to identify agents and retrieve their properties from an external system instead.

There can be more than one agent for each activity and one agent can be responsible for more than one activity or entity, using the relations defined in the following sections.

AgentType

Literal	Comment
Person	person agents are people, e.g. described by their name or their official position
Organization	an organization is a social or legal institution, e.g. an institute, a consortium, a project
SoftwareAgent	a software agent is running software, e.g. a cron job or a trigger

Table 6: Enumeration of Agent types.

2.4.2 WasAssociatedWith class

An activity **association** is an assignment of responsibility to an agent for an activity, indicating that the agent had a role in the activity (W3C PROV-DM §5.3.3).

Association is implemented in the model by a class *WasAssociatedWith* that connects *Activity* to *Agent* and contains the attributes in Table 7.

For example, the agent "Max Smith" wasAssociatedWith the activity "observation" with the role "Observer".

WasAssociatedWith

Attribute	Data type	Description
role	string	function of the agent with respect to the activity

Table 7: Attributes of *WasAssociatedWith* relation class.

2.4.3 WasAttributedTo class

Attribution is the ascribing of an entity to an agent. When an entity is attributed to an agent, this entity was generated by some unspecified activity that in turn was associated to the agent. Thus, this relation is generally useful when the activity is not known, or irrelevant (W3C PROV-DM §5.3.2).

Attribution is implemented in the model by a class *WasAttributedTo* that connects *Entity* to *Agent* and contains the attributes in Table 8.

For example, the entity "science_image.fits" wasAttributedTo the agent "observatory".

WasAttributedTo

Attribute	Data type	Description
role	string	function of the agent with respect to the entity

Table 8: Attributes of *WasAttributedTo* relation class.

2.4.4 Agent roles

Agents may play a specific role with respect to an activity or an entity. The `role` attribute should be specified whenever it is known.

Roles in relations to Agent are free text attributes, but if one of the terms in Table 9 applies, it should be used.

Agent roles

Role	Comment
Author	the agent is an author of this entity (e.g. article, software, proposal)
Contributor	the agent helped in the creation of this entity
Coordinator	the agent was coordinating a specific activity
Creator	the agent created this entity (creators of articles or software are rather called "author")
Curator	the agent curated this entity
Editor	the agent was responsible for editing this entity
Funder	the agent provided financial support for this activity or the creation of the entity
Investigator	the agent is responsible for the scientific goals of this activity
Observer	the agent was observing and is associated to a specific "observation" activity or responsible for observing a specific entity
Operator	the agent was an operator for a specific activity
Provider	the agent provided this entity
Publisher	the agent published this entity

Table 9: Terms applicable as agent roles.

2.5 Description classes

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. We therefore separate the descriptions of activities from the actual processes and introduce an *ActivityDescription* class (Section 2.5.1). Likewise, we also apply the same pattern for *Entity* and add an *EntityDescription* class (Section 2.5.2).

Defining such descriptions allows them to be predefined and reused, which is less redundant when exposing the provenance of a series of tasks of the same type. Providing detailed descriptions to activities and entities help assess the quality and reliability of the processes executed (see goal C in Section 1.1).

Figure 5 shows the class diagram part focused on the description classes.

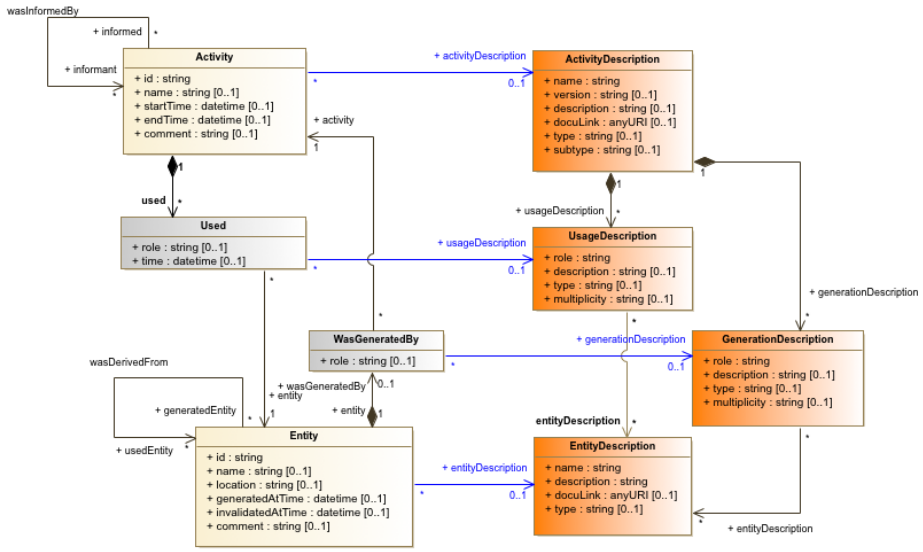

Figure 5: Partial class diagram focused on description classes.

2.5.1 ActivityDescription class

How an activity works internally is further explained by information contained in the *ActivityDescription* class. An activity is then a concrete case (instance) with a given start and stop time, and it refers to a description for further information.

ActivityDescription is directly attached to *Activity* and can thus be seen as a list of attributes that can be known before an *Activity* instance is created.

There must be exactly zero or one *ActivityDescription* per *Activity*. Note that if an instance of *Used*/*WasGeneratedBy*/*Entity* is linked to its corresponding description (see next sections), then there must exist a link between the corresponding activity and activity description.

ActivityDescription

Attribute	Data type	Description
name	string	a human-readable name
version	string	a version number, if applicable (e.g. for the code used)
description	string	additional free text description for the activity
doculink	anyURI	link to further documentation on this activity, e.g. a paper, the source code in a version control system etc.
type	string	type of the activity
subtype	string	more specific subtype of the activity

Table 10: Attributes of the *ActivityDescription* class. Attributes in **bold** must not be null.

The activity `type` is a free text attribute, but if one of the terms in Table 11 applies, it should be used. The activity `subtype` is a free text attribute to be used internally by the project that defined *ActivityDescription* instances (e.g. mosaicing, denoising, photometric calibration, cross correlation).

ActivityDescription types

Type	Comment
Observation	Active acquisition of information on a phenomenon
Simulation	Generation of data through a computational process
Reduction	Transformation of digital information into a corrected, ordered, and simplified form
Calibration	Transformation and comparison of measurement values with respect to a calibration standard of known accuracy
Reconstruction	Estimation of physical properties using indirect information
Selection	Application of filters or criteria to select partial information
Analysis	Process of inspecting, cleansing, transforming, and modeling data with the goal of discovering useful information, informing conclusions, and supporting decision-making

Table 11: Terms applicable as activity types.

2.5.2 EntityDescription class

EntityDescription

Attribute	Data type	Description
name	string	a human-readable name for the entity description
description	string	a descriptive text for this kind of entity
doculink	anyURI	link to more documentation
type	string	type of the entity

Table 12: Attributes of the *EntityDescription* class.

The *EntityDescription* class is meant to store descriptive information for different categories of entities. It contains information that is known before an *Entity* instance is created. The *EntityDescription* general attributes are summarized in Table 12.

For example, a specific category of entities in a project may be defined in details in a document or on a webpage (e.g. a CTA DL3 file, a CCD device, a photographic plate).

The entity `type` is a free text attribute, that contains the general category of the entity, e.g. if it is data, a document, a vizualization, a device...

The *EntityDescription* class should not contain information about the usage of the data, in particular, it generally tells nothing about them being used as input or generated as output. This kind of information should be provided by the relations (and their descriptions) between activities and entities (see Sections 2.3 and 2.5.3).

2.5.3 UsageDescription and GenerationDescription classes

In order to describe more precisely an activity, the expected inputs and outputs of this activity should be specified.

We introduce the *UsageDescription* and the *GenerationDescription* classes, that are meant to store the information about the usage or generation of entities that is known before an activity instance is executed, i.e. what we expect to store in the *Used* and *WasGeneratedBy* relations (see 2.3). Instances of *Used* (respectively *WasGeneratedBy*) may thus point to an instance of *UsageDescription* (respectively *GenerationDescription*).

If a *UsageDescription* (respectively *GenerationDescription*) instance is defined, the `role` attribute of the related *Used* (respectively *WasGeneratedBy*) instances must match the `role` attribute of this *UsageDescription* (respectively *GenerationDescription*) instance.

UsageDescription

Attribute	Data type	Description
role	string	function of the entity with respect to the activity
description	string	a descriptive text for this kind of usage
type	string	type of relation, see Section 2.5.4
multiplicity	string	Number of expected input entities to be used with the given role. The string "unbounded" indicates that the number of input entities can be more than 1.

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

GenerationDescription

Attribute	Data type	Description
role	string	function of the entity with respect to the activity
description	string	a descriptive text for this kind of generation
type	string	type of relation, see section 2.5.4
multiplicity	string	Number of expected output entities that will be generated with the given role. The string "unbounded" indicates that the number of output entities can be more than 1.

Table 14: Attributes of the *GenerationDescription* class. Attributes in **bold** must not be null.

A `multiplicity` attribute should be specified to indicate the number of entities expected to share the same role for a given ActivityDescription instance, e.g. in the case of the stacking of images, several images are expected with the same input role (`multiplicity=*`).

When related to the *UsageDescription* or *GenerationDescription*, the attributes of *EntityDescription* (see Section 2.5.2) help to describe the category of entities expected as an input or an output in an activity. For example: when the input bias files are expected to be in FITS format (relation to a *DatasetDescription* with `contentType`=application/fits), or the red, green and blue channel images are expected to be in PNG or JPEG format.

2.5.4 Types of Usage and Generation

The typing of those relations is particularly needed to enable quality assessment and identification of error sources in the process (see goals C and D in Section 1.1), so as to facilitate the exploration of provenance information.

The type of usage or generation is an optional attribute. It is a free text attribute, but if one of the terms in Table 15 applies, it should be used.

Type	Description
Main	main input or output entities of the activity, i.e. strictly necessary, and the primary objective of the activity.
Calibration	usage of an entity to calibrate another entity.
Preview	generation of a quick representation of an entity.
Setup	usage of an entity as configuration information, see also Section 2.7.1
Quality	generation of information that helps to assess the quality of the activity results, example: errors, warnings, flags, percentage of overexposed pixels, ...
Log	generation of logging information
Context	contextual information that influences the activity, but for which there are no or little control at the moment of its execution, examples: temperature, wind, conditions of observation, execution platform, operating system, instrumental context, ...

Table 15: Terms applicable as usage or generation type.

The type "Main" indicates the main input and output entities of an activity. It should help to provide the minimum relevant data flow to the initial entity or activity, i.e. to find the most relevant progenitors.

2.6 Specific types of Entity classes

Entity and *EntityDescription* classes carry the minimum metadata that can apply to any kind of entity without specifying the nature or the structure of the content of the entity. In some cases, the structure of the content is relevant information to assess the usefulness of the entity, in particular for datasets.

In some other cases, the content itself of an entity is relevant information to assess the usefulness of the related entities or activities. Such content must then be expose as properly described values.

In astronomy and the VO, we thus define two main types of entity classes:

- **Dataset**: a dataset is a resource which encodes data in a defined structure. It is generally a file or a set of files which are considered to be a single deliverable. The content may be e.g. a cube, an image, a table, a list.
- **Value**: a value is an atomic piece of data with a given value type (e.g. a data type such as boolean, integer, real, string).

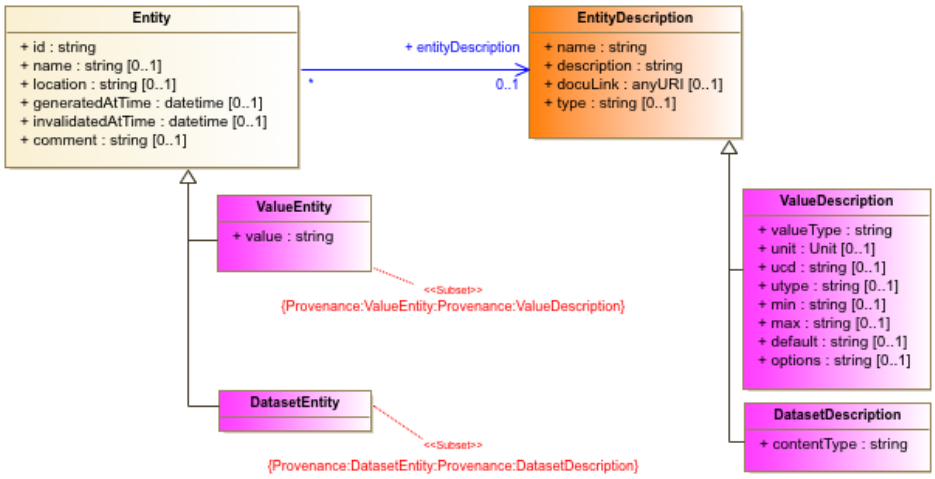

Figure 6: Partial class diagram focused on Specific types of *Entity* classes.

As shown in Figure 6, the entity description classes for both *ValueEntity* and *DatasetEntity* are subsetted respectively as *ValueDescription* or *DatasetDescription*.

We anticipate that more specific categories of entities can be defined by the projects (for example, a device, a document, a vizualization...). The `type` attribute of the *EntityDescription* class should be used to differentiate the different categories of entities.

2.6.1 DatasetEntity and DatasetDescription classes

The handling of datasets is implemented in the model by a *DatasetEntity* class. A corresponding *DatasetDescription* class contains a `contentType` attribute that must not be null (see Table 16).

The `contentType` indicates the MIME-type or format of a dataset, or a more precise structure, following the definition of the attribute `access_format` defined in ObsCoreDM (Louys and Tody et al. (2017), Section 4.7).

DatasetDescription

Attribute	Data type	Description
contentType	string	MIME-type or format of the dataset

Table 16: Attributes of the *DatasetDescription* class. The class also inherits the attributes of *EntityDescription* listed in Table 12. Attributes in **bold** must not be null.

2.6.2 ValueEntity and ValueDescription classes

The handling of values is implemented in the model by a *ValueEntity* class that contains a `value` attribute. A corresponding *ValueDescription* class contains attributes commonly used in the VO to qualify values. Those attributes are listed in Table 18.

ValueEntity

Attribute	Data type	Description
value	string	the value of the entity. If a corresponding *ValueDescription*.`valueType` attribute is set, the value string can be interpreted by this `valueType`.

Table 17: Attributes of the *ValueEntity* class. The class also inherits the attributes of *EntityDescription* listed in Table 12. Attributes in **bold** must not be null.

ValueDescription

Attribute	Data type	Description
valueType	string	value types of the VODML ivoa base model, see Lemson and Laurino et al. (2018)
unit	Unit	physical unit, see C.1.1 and Derriere and Gray et al. (2014) for recommended unit representation
ucd	string	Unified Content Descriptor, supplying a standardized classification of the physical quantity, see Martinez and Louys et al. (2018)
utype	string	Utype, meant to express the role of the value in the context of an external data model, see Graham and Demleitner et al. (2013)
min	string	minimum value as a string whose value can be interpreted by the `valueType` attribute
max	string	maximum value as a string whose value can be interpreted by the `valueType` attribute
options	string	comma separated list of possible values
default	string	the default value of the value entity as a string whose value can be interpreted by the `valueType` attribute

Table 18: Attributes of the *ValueDescription* class. The class also inherits the attributes of *EntityDescription* listed in Table 12. Attributes in **bold** must not be null.

2.7 Activity configuration

Configuring an activity is the way to set parameters so that the activity occurs in the desired conditions.

In some cases developed in Section 1.1 (goals C and D in particular), configuration information is relevant to assess the quality and reliability of an activity or an entity, and to identify the location of configuration errors in a processing. It also facilitates the re-execution of an activity (reproducibility).

Configuration information may be carried by entities using the core features, where an entity (e.g. *ValueEntity* and *DatasetEntity* instances) is referenced in *Used* relations with a given `role` and `type`="setup". With this solution, the configuration information is independent from the activity and can be generated and used as any entity.

The data model also provides a specialized *ActivityConfiguration* package to directly attach configuration information to an activity. This package is composed of a *WasConfiguredBy* relation connecting *Parameter* and *ConfigFile* classes with the *Activity* class (see 2.7.1). With this solution the configuration information is independent from the entities, and seen as part of the activity.

2.7.1 Overview of the ActivityConfiguration package

As shown in Figure 7 the *ActivityConfiguration* package contains two classes for the execution side: *Parameter* and *ConfigFile* which are connected to an *Activity* instance via the *WasConfiguredBy* association class. An *Activity* may thus be configured by a set of *Parameter* instances, or by *ConfigFile* instances containing a list of (key,value) pairs, or by a combination of both.

The corresponding description classes, *ParameterDescription* and *ConfigFileDescription*, are both defined in the context of the description of an activity. There can be several instances of a *Parameter* (respectively *ConfigFile*) that are described by the same instance of *ParameterDescription* (respectively *ConfigFileDescription*).

2.7.2 Parameter and ParameterDescription classes

The *Parameter* class contains a `value` and a `name` attribute that must be set (Table 19).

The *ParameterDescription* class describes the parameter `value` attribute similarly to the *ValueEntity* and *ValueDescription* classes. Those attributes are listed in Table 20.

If a *ParameterDescription* instance is defined, the `name` attribute of the related *Parameter* instances must match the `name` attribute of this *ParameterDescription* instance.

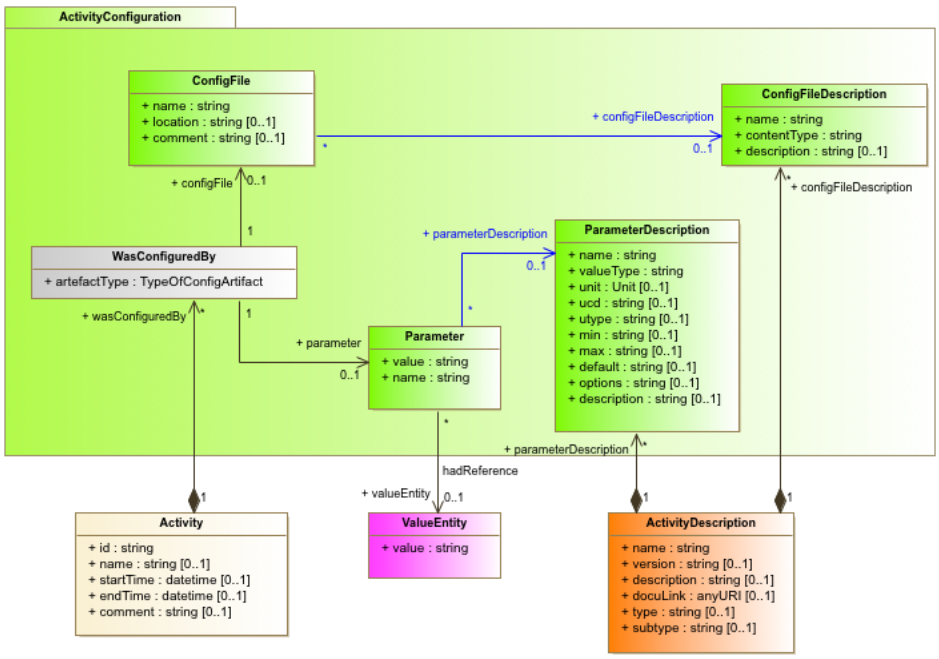

Figure 7: Partial class diagram focused on the *ActivityConfiguration* package. The *Parameter* and *ConfigFile* classes provide configuration information for an *Activity* instance. The right side of the diagram shows the descriptions, where an *ActivityDescription* class is bound with the *ParameterDescription* and *ConfigFileDescription* classes.

Parameter

Attribute	Data type	Description
name	string	name of the parameter
value	string	the value of the parameter. If a corresponding *ParameterDescription*.`valueType` attribute is set, the value string can be interpreted by this `valueType`.

Table 19: Attributes of the *Parameter* class. Attributes in **bold** must not be null.

2.7.3 ConfigFile and ConfigFileDescription classes

The *ConfigFile* is a text file, where key value pairs are listed as parameters for running an activity. It contains a `location` and a `name` that must be set, and a `comment` attribute (Table 21).

The *ConfigFileDescription* class indicates the format in which the list is

ParameterDescription

Attribute	Data type	Description
name	string	name of the parameter
valueType	string	value types of the VODML ivoa base model, see Lemson and Laurino et al. (2018)
description	string	a descriptive text for the parameter
unit	Unit	physical unit, see C.1.1 and Derriere and Gray et al. (2014) for recommended unit representation
ucd	string	Unified Content Descriptor, supplying a standardized classification of the physical quantity, see Martinez and Louys et al. (2018)
utype	string	Utype, meant to express the role of the parameter in the context of an external data model, see Graham and Demleitner et al. (2013)
min	string	minimum value as a string whose value can be interpreted by the `valueType` attribute
max	string	maximum value as a string whose value can be interpreted by the `valueType` attribute
options	string	comma separated list of possible values
default	string	the default value of the parameter as a string whose value can be interpreted by the `valueType` attribute

Table 20: Attributes of the *ParameterDescription* class. Attributes in **bold** must not be null.

provided in a `contentType` attribute (see Table 22).

If a *ConfigFileDescription* instance is defined, the `name` attribute of the related *ConfigFile* instances must match the `name` attribute of this *ConfigFileDescription* instance.

2.7.4 Relations with Activity class

The relation of *Parameter* and *ConfigFile* to *Activity* is formalized by a *WasConfiguredBy* class. There must be exactly one instance connected to a *WasConfiguredBy* instance, either a *Parameter* instance or a *ConfigFile*

ConfigFile

Attribute	Data type	Description
name	string	a human-readable name for the config file
location	string	a path to the config file, e.g. a URL
comment	string	text containing comments on the config file

Table 21: Attributes of the *ConfigFile* class. Attributes in **bold** must not be null.

ConfigFileDescription

Attribute	Data type	Description
name	string	a human-readable name for the config file
contentType	string	MIME-type or format of the dataset
description	string	a descriptive text for the config file

Table 22: Attributes of the *ConfigFileDescription* class. Attributes in **bold** must not be null.

instance. The *WasConfiguredBy* class contains the attribute `artefactType` to indicate the type of class pointed by the *WasConfiguredBy* instance (see Table 23).

The life cycle of a *Parameter* instance (respectively *ConfigFile* instance) is the one of the corresponding *Activity* instance. The life cycle of a *ParameterDescription* instance (respectively *ConfigFileDescription* instance) is the one of the corresponding *ActivityDescription* instance. This means that when an activity is deleted from the provenance repository, its parameters and config files also disappear.

Several activities launched with various possible values for a parameter share the same *ParameterDescription* instance. For instance, a cube analysis activity with a parameter "nbofChannels" will point to the corresponding instance of *ParameterDescription* (`name` = "nbofChannels", `ucd` = "meta.number", `unit` = NULL, `description` = "Nb of channel used for segmentation").

Similarly, we can foresee a number of different *ConfigFile* instances used for various instances of an *Activity*, which rely on the same *ConfigFileDescription* instance bound to the corresponding *ActivityDescription* instance.

The *Parameter* instance may refer to a *ValueEntity* instance using a

WasConfiguredBy

Attribute	Data type	Description
artefactType	TypeOfConfigArtefact	string that takes the value "Parameter" or "ConfigFile" to indicate the type of class pointed by the *WasConfiguredBy* instance.

Table 23: Attributes of the *WasConfiguredBy* class. Attributes in **bold** must not be null.

hadReference which gives the origin of this value.

3 Full class diagram

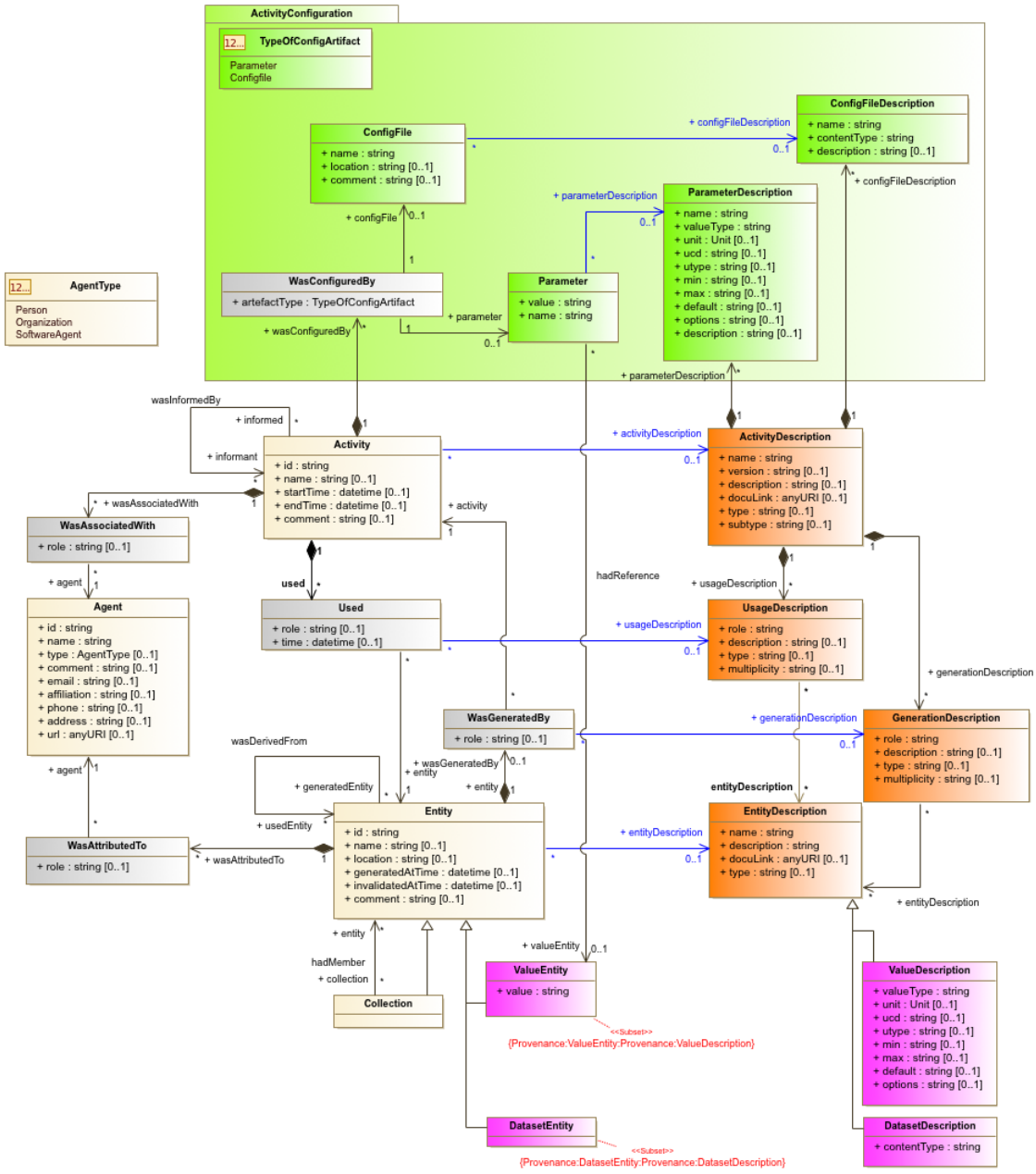

Figure 8: Full class diagram of the IVOA Provenance Data Model.

Appendix A Changes from Previous Versions

A.1 Changes from PR-ProvenanceDM-1.0-20181015

- Rewording of the use cases in the Goal section.
- Requirements divided into Model Requirements and Best Practices.
- Restruturation of Section 2, removing the separation of the Core Model and the Extended Model. An overview is now given in 2.1, and each following subsection corresponds to a feature of the model, in relation with the goals. Each subsubsection is an element of the model presented in Figure 3.
- *WasInformedBy* and *WasDerivedFrom* are optional and included in section 2.3 on *Entity-Activity* relations. They are simple relations.
- Addition of *ValueEntity* and *DatasetEntity* (with their description classes) as specific types of Entity classes (section 2.6).
- Grouping of the *Parameter-ParameterDescription* classes as an optional *ActivityConfiguration* package that also includes config files.
- Modification of class attributes:
  - `annotation` → `comment` (`annotation` → `description` in Description classes)
  - `creationTime` → `generatedAtTime` (W3C PROV term)
  - `destructionTime` → `invalidatedAtTime` (W3C PROV term)
  - added: *Agent*.`url`
  - removed: *WasGeneratedBy*.`time`, *Entity*.`rights`, *Activity*.`status`, *ParameterDescription*.`xtype`, *ParameterDescription*.`arraysize`.
- Datatype of a value is replaced by a valueType (see VO-DML document).
- Only attributes are shown in tables, relations between classes are shown in diagrams.
- Lists of terms are explicitely given for *Agent* roles, *ActivityDescription* types, *UsageDescription*/*GenerationDescription* types.
- The section on serializations was removed (a dedicated document will be proposed). References to serialization in the text were removed, as well as related sentences, and Appendix A.

- The section on accessing provenance information was removed (it was already reduced to a paragraph referencing external documents in preparation).
- Appendix B on Links to other data models was removed.
- Modeling Conventions and Data Types were added as appendices (taken from STC/Meas document).

A.2 Changes from WD-ProvenanceDM-1.0-20180530

- Separate core model (W3C only) and extended model (IVOA Provenance DM).
- Add definitions for specialised entities, including all the description classes and parameters.
- Add definitions for specialised relations.
- Updated serialization of the description classes for web services.

A.3 Changes from WD-ProvenanceDM-1.0-20170921

- Moved ProvDAL (now ProvSAP) to separate document.
- Moved ProvTAP section and full definition of VOTable serialisation to separate ProvTAP document.
- Moved chapter 6 with use cases and "How to use the data model" to separate Implementation Note (Servillat and Riebe et al., 2017).
- Moved section on links to other data model into appendix.
- ParameterDescription: Added attributes `xtype` and `arraysize`.
- Agent.roles: Removed "PI" alternative to "coordinator".
- Use values of RightsType of DatasetDM, public, secure, proprietary, for `Entity.rights`.
- Minor corrections in HiPS use case, and tables in TAP schema.
- Minor correction in role names for hadStep/hadMember relationship.
- Modified text on Parameters
- Rename 3.4 section to Serialization of description classes for web services
- Modified text on W3C serialization
- add `location` and `value` attributes to *Entity*

A.4 Changes from WD-ProvenanceDM-1.0-20161121

- New appendix for PROV-VOTable/TAP SCHEMA tables added
- Corrected and extended attribute tables and mapping tables for links with DatasetDM and SimDM.
- Restructured Accessing provenance section by splitting it in two: Section for explaining the different serialization formats and differences to W3C serializations, Section "Accessing provenance information"for describing the access protocols ProvDAL and ProvTAP.
- Removed discussion section, since now all the topics are addressed in the main text.
- Added paragraph on how to use the model in Section 6.
- Shortened serialization examples, partially moved them to appendix.
- Added paragraph on VOSI interface.
- Added a proposed serialization of description classes.
- Modified text on the content of EntityDescription, now seen as Entity attributes known before the Entity instance exists.
- Renamed Section 6 to stress that it explains applications of the model (use cases); implementation details and code examples can be found in Implementation Note (Servillat and Riebe et al., 2017).
- Complete rewrite of the ProvDAL section in Section "Accessing provenance information";new parameters, new figure and examples added.
- Added additional figure for entity-activity relations.
- Moved the figure showing relations between Provenance.Agent and Dataset.Party into the Section on data model links.
- Extended the entity role examples in table `tab:entity-roles`.
- Added links to provn and votable-serialization for HiPS-use case, added first part of provn as example in the HiPS-use case section.
- More explanations on links to data models in a dedicated section, introduced subsections, added table with SimDM-mapping.
- Moved detailed implementation section from appendix to a separate document (implementation note), shortened the use cases & implementation section.

- Attribute/class updates:
  - Added attribute *votype* to *Activity*, can be used for ActivityFlows
  - Added attribute *time* to *Used* and *WasGeneratedBy*
  - Added optional attributes *Entity.creationTime* and *EntityDescription.category*
  - Added optional attributes *Parameter.min*, *Parameter.max*, *Parameter.options*
  - Removed the obscore/dataset attributes from EntityDescription, since they are specific for observations only and are not applicable to configuration entities etc.
  - Use voprov:type and voprov:role in Table on Agent roles, i.e. replaced prov:person by Individual and prov:organization by Organization.
  - Renamed *label* attribute to *name* everywhere, for more consistency with SimDM naming scheme (*label* is reserved there for SKOS labels).
  - Renamed attribute *Entity.access* to *Entity.rights* for more consistency with DatasetDM etc.
  - Avoid double-meaning of *description* (as reference and free-text description) by renaming the free-text description to *annotation*. Mark description-references with arrows in attribute tables.
  - Applied similar naming scheme to *Parameter* and *ParameterDescription*-classes
  - Renamed *docuLink* to *doculink*
  - Corrected attribute names in Table on mapping to DatasetDM.

Appendix B Modeling Conventions

This model follows the VO-DML modeling practices, however, the UML representations may vary depending on the tool used. Below we describe the graphical representation of the modeling concepts and relations.

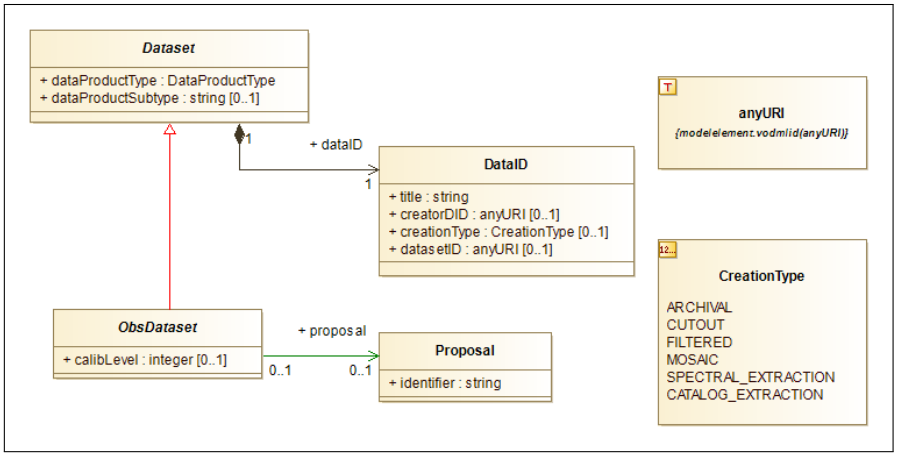

Figure 9: Notation example diagram

B.1 Class

Classes are represented by a plain box. The class name is annotated in the top window, abstract classes use italic typeface. Attributes, if any, are listed in the lower panel. Attributes may only be of primitive type (real, string, etc), a defined DataType, or an Enumeration type. Relationships to other objects are defined via the composition and reference relation arrows.

B.2 DataType

DataTypes are represented by a box shape similar to Class, but annotated with a "T" symbol in the top left corner.

B.3 Enumerations

Enumerations are represented by a box shape similar to Class, but annotated with a "1,2.." symbol in the top left corner. Enumeration Literals (possible values) are listed below the enumeration class name.

B.4 Generalization

Generalizations are represented by a line (shown in red in Figure 9), with open triangle at the end of the source, or more general, object.

B.5 Composition

The composition relation is indicated by a line with a solid diamond attached to the containing object, and an arrow pointing to the object being contained. The composition relation is very tight, where the container is responsible for the creation and existence of the target. Any object may be in no more than one composition relation with any container. The attribute name for the composition relation is annotated at the destination of the relation (e.g. "+ dataID"). This is typically a lower-cased version of the destination class name, but this is not required.

B.6 Reference

The reference relation is indicated by a line (shown in green in Figure 9), with an arrow pointing to the object being referenced. The reference relation is much looser than composition, the container has no ownership of the target, but merely holds a pointer, or other indirect connection to it. The attribute name is annotated at the destination of the relation (e.g. "+ proposal"). This is typically a lower-cased version of the destination class name, but may be another name indicating the role that the class is playing in this context.

B.7 Multiplicity

All attributes and relations have a multiplicity associated with them. For attributes, the multiplicity is contained within brackets just after the attribute name. If no bracket is displayed, this is equivalent to '[1]'.

- 1 = one and only one value must be provided.
- 0..1 = zero or one value may be provided.
- * = zero or more values may be provided (open ended).

Appendix C Data Types

C.1 Base Data Types

Provides a set of standardized primitive data types as well as types for representing quantities (values with associated units). We provide a diagram of the model here, and refer the reader to Section 5 of the VO-DML modeling specification document (Lemson and Laurino et al., 2018) for more information.

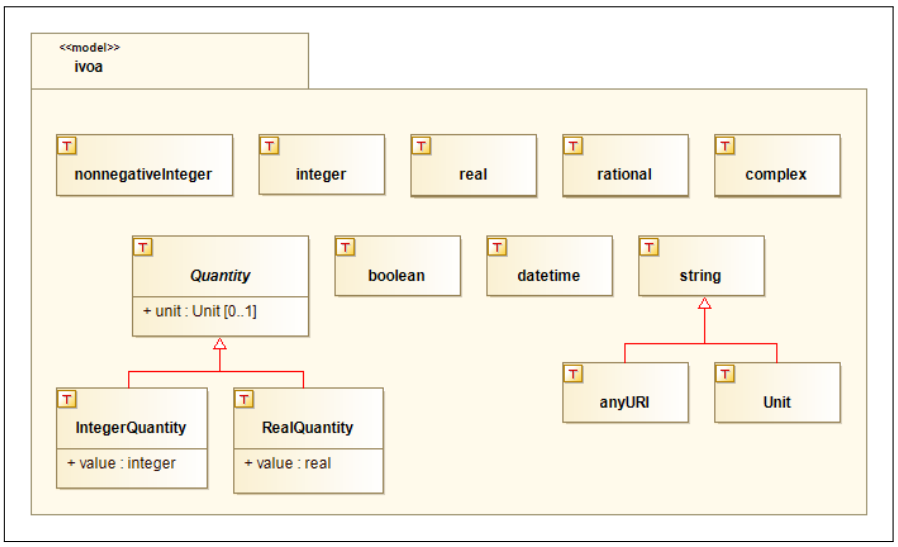

Figure 10: Base Data Types

C.1.1 Units

This model requires the use of the IVOA VOUnits Standard (Derriere and Gray et al., 2014) for representing units of physical quantities. This standard reconciles common practices and current standards for use within the IVOA community.

C.1.2 Dates

The 'datetime' datatype is for expressing date-time values. The string representation of a datetime value should follow the FITS convention for representing dates. The FITS standard is effectively ISO8601 format without the "Z" tag to indicate UTC (YYYY-MM-DDThh:mm:ss). Values are nominally expressed in UTC.

List of Figures

List of Tables

Bibliography

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.
https://www.w3.org/TR/2013/REC-prov-dm-20130430/

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

Derriere, S., Gray, N., Demleitner, M., Louys, M. and Ochsenbein, F. (2014), 'Units in the VO Version 1.0', IVOA Recommendation 23 May 2014, arXiv:1509.07267.
http://doi.org/10.5479/ADS/bib/2014ivoa.spec.0523D

Graham, M., Demleitner, M., Dowler, P., Fernique, P., Laurino, O., Lemson, G., Louys, M. and Salgado, J. (2013), 'UTypes: current usages and practices in the IVOA', IVOA Note.
http://www.ivoa.net/documents/Notes/UTypesUsage

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

Lemson, G., Laurino, O., Bourges, L., Cresitello-Dittmar, M., Demleitner, M., Donaldson, T., Dowler, P., Graham, M., Gray, N., Michel, L. and Salgado, J. (2018), 'VO-DML: a consistent modeling language for IVOA data models Version 1.0', IVOA Recommendation 10 September 2018.
https://ui.adsabs.harvard.edu/abs/2018ivoa.spec.0910L

Louys, M., Richards, A., Bonnarel, F., Micol, A., Chilingarian, I., McDowell, J. and IVOA Data Model Working Group (2008), 'Data Model for Astronomical DataSet Characterisation Version 1.13', IVOA Recommendation 25 March 2008, arXiv:1111.2281.
http://doi.org/10.5479/ADS/bib/2008ivoa.spec.0325L

Louys, M., Tody, D., Dowler, P., Durand, D., Michel, L., Bonnarel, F., Micol, A. and IVOA DataModel Working Group (2017), 'Observation Data Model Core Components, its Implementation in the Table Access

Protocol Version 1.1', IVOA Recommendation 09 May 2017.
http://doi.org/10.5479/ADS/bib/2017ivoa.spec.0509L

Martinez, A. P., Louys, M., Cecconi, B., Derriere, S., Ochsenbein, F. and IVOA Semantic Working Group (2018), 'The UCD1+ controlled vocabulary Version 1.3 Version 1.3', IVOA Recommendation 27 May 2018.
http://doi.org/10.5479/ADS/bib/2018ivoa.spec.0527M

McDowell, J., Tody, D., Budavari, T., Dolensky, M., Kamp, I., McCusker, K., Protopapas, P., Rots, A., Thompson, R., Valdes, F., Skoda, P., IVOA Data Access Layer and Data Model Working Groups (2007), 'IVOA Spectral Data Model Version 1.03', IVOA Recommendation 29 October 2007.
http://doi.org/10.5479/ADS/bib/2007ivoa.spec.1029M

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/

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

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L. B., Bourne, P. E., Bouwman, J., Brookes, A. J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C. T., Finkers, R., Gonzalez-Beltran, A., Gray, A. J. G., Groth, P., Goble, C., Grethe, J. S., Heringa, J., 't Hoen, P. A. C., Hooft, R., Kuhn, T., Kok, R., Kok, J., Lusher, S. J., Martone, M. E., Mons, A., Packer, A. L., Persson, B., Rocca-Serra, P., Roos, M., van Schaik, R., Sansone, S.-A., Schultes, E., Sengstag, T., Slater, T., Strawn, G., Swertz, M. A., Thompson, M., van der Lei, J., van Mulligen, E., Velterop, J., Waagmeester, A., Wittenburg, P., Wolstencroft, K., Zhao, J. and Mons, B. (2016), 'The fair guiding principles for scientific data management and stewardship', *Scientific Data* **3**, 160018 EP –.
http://dx.doi.org/10.1038/sdata.2016.18