Provenance Data Model
Version 0.1

IVOA Working Draft 7 May 2014

Working Group:
    IVOA Data Model
This version:
    http://www.ivoa.net/Documents/ProvDM-20140507
Latest version:
    http://www.ivoa.net/Documents/latest/latest-version-name
Previous versions:
Authors (possibly):
    Kristin Riebe
    Florian Rothmaier
    Markus Demleitner
    Gerard Lemson
    Adrian Partl
    Jochen Klar
    Harry Enke
    Mireille Louys

Abstract

Type your abstract here

Status of this Document

Acknowledgments

This document has been developed in part with support from the German Astrophysical Virtual Observatory (BMBF Bewilligungsnummer 05A08VHA).

Conformance-related definitions

The words "MUST ", "SHOULD", "MAY", "RECOMMENDED", and "OPTIONAL" (in upper or lower case) used in this document are to be interpreted as described in the IETF standard RFC 2119 *[std:RFC2119]*.

Contents

1. Introduction

In this document, we discuss a draft for an IVOA standard data model for describing the provenance of data. We focus here on observational data, since provenance for simulated data is already covered by SimDM *[std:SimDM]*. This draft is at a very preliminary state and may change at any time.

1.1. Requirements for Provenance and Use Cases

A provenance data model should provide solutions to the following tasks:

- **A: Tracking the production history**
  Find out which steps were taken to produce this piece of data/image and which methods/tools/software was involved. Track the history back to the raw data files/raw images.
  Examples:
    - Is the image from catalogue xxx already calibrated? What about dark field subtraction? Were foreground stars removed? Which technique was used?
    - Is the background noise of atmospheric muons still present in my neutrino data sample?
  We don't go so far as to consider easy reproducability as a use case -- this would be too ambitious. But at least the major steps undertaken to create a piece of data should be recoverable.
- **B: Attribution and further information**

Find the people involved in the production, the people/organizations/institutes that need to be cited or can be asked for more information
  - I want to use an image for my own work - who was involved in creating it? Whom do I need to cite or who can I contact to get this information?
  - I have a question about column xxx in the data table. Who can I ask about that?
- **C: Aid in debugging**
  Find possible error sources.
  - I found something strange in an image. Where does the image come from? Which instrument was used, with which characteristics etc.? Was there anything strange noted when the image was taken?
  - Which pipeline version was used -- the old one with a known bug for treating bright objects or a newer version?
  - This light curve doesn't look quite right. How was the photometry determined for each data point?
- **D: Quality assessment**
  Judge the quality of an observation, production step or data set.
  - Since wrong calibration images may increase the number of artifacts on an image rather than removing them, the knowledge about the calibration image set will help to assess the quality of the calibrated image.
- **E: Search in structured provenance metadata**
  Find all images produced by a certain processing step and similar tasks.
  - Give me more images that were produced using the same pipeline.
  - Give me an overview on all images reduced with the same calibration data set.
  - Are there any more images attributed to this observer?
  - Which images of the crab nebula are of good quality and were produced within the last 10 years by someone not from ESO or NASA?

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

1.2. Previous efforts on provenance modelling

Provenance in the IVOA [TODO]

Provenance was already discussed since the early days of the IVOA, but previous efforts from the IVOA community unfortunately are mostly not well documented or we were not able to find them. There exists a vocabulary list by Arnold Rots, we got a provenance diagram from a talk by Francois Bonnarell, and there exist workflows to track provenance for specific projects (e.g. CTA), but this is all very specific and too detailed for a general provenance model. We therefore went a step back to start with a more abstract core model as discussed in this document.

TODO: give more specific references, check other data models and where they touch provenance

Open Provenance Model (OPM) and W3C PROV-DM

Other communities have put considerable efforts into establishing a provenance model. The Provenance Challenge series (2006 - 2010) was a community effort to achieve inter-operability between different representations of provenance in scientific workflows. It resulted in the Open Provenance Model (*[OPM]*), and some members of that challenge continued their work by joining the W3C Provenance Working Group. The W3C Provenance Data Model was released as Recommendation in 2013. The cores of OPM and W3C are in principle the same, therefore we describe here briefly the basic concepts of the W3C Provenance Data Model (PROV-DM, see *[std:W3CProvDM]*) in this section.

OPM was designed to be applicable to anything, scientific data as well as cars or immaterial things like decisions. With W3C, this becomes more focused on the web. Nevertheless, the core concepts are still fairly general and should be useful for astronomical data sets as well, though we could aim at making it more specific and less abstract for our purposes.

The elements of a provenance model can be expressed as a directed graph to capture the causal dependencies. In W3C, there are three core components, i.e. nodes of the graph: (see Section 1+2 and Fig. 1 in *[std:W3CProvDM]* for the detailed specification, which we reproduce here only partially):

- Entity: a thing, at a certain state (car, paper, data set, idea)
- Activity: an action/process or series of actions, occurs over a period of time, performed on or caused by entities, usually results in new entities
- Agent: executes/controls an activity

There are also different possible relations between these components. Those important for astronomical data are:

- Generation: a new entitiy is generated by an activity (output_data wasGeneratedFrom activity)
- Usage: an entity is used by an activity (activity used input_data)
- Association: agents have responsibility for an activity (agent wasAssociatedWith activity)

There is one more relation that could be interesting for us:

- Attribution: an entitiy can be attributed to an agent (entity wasAttributedTo agent)

The attributed-to-agent may be different from the agent that is associated with the activity that created an entity. The following figure summarizes the (core) classes and relations of the W3C provenance model that are interesting for us:

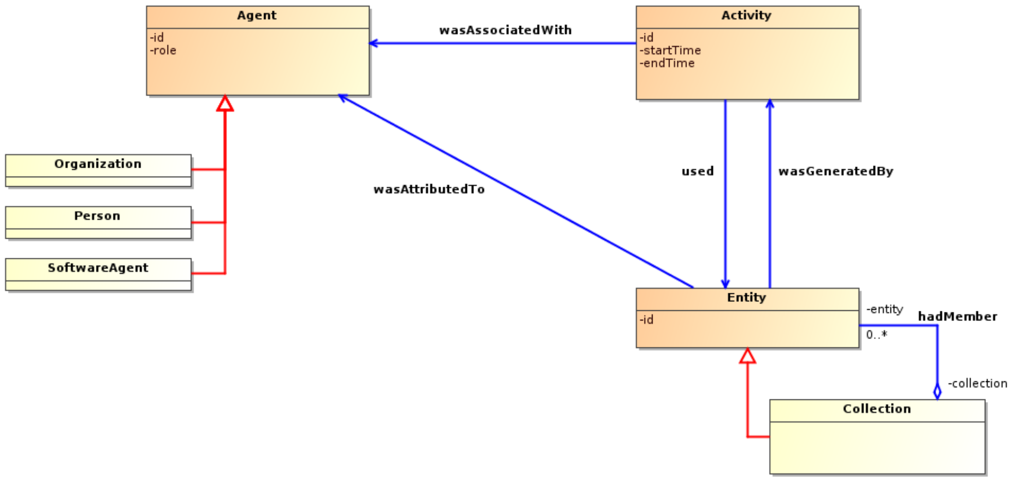

Figure 1: The main classes and relations of the W3C Prov-DM, that could be useful for the VO.

The W3C PROV-DM further specifies a derivation relation, which directly links a derived entity to its predecessor. This was introduced to make the link between entities more explicite. If activity a used entities e1 and e2 as input and produced e3 as output, then it is not clear, if e3 is a derivative (or, in W3C terms, "wasDerivedFrom") e1 or e2 or both or neither. e1 and e2 could have been only parameter files for an observation. However, for the IVOA provenance model we are fairly sure that this is not needed. Instead, at the moment we favour adding a *role* to each dataItem or entity, which can explain in more detail in which way the entity was used. We will discuss later on an approach using the additional Method-class as prototype or template for each activity. The types of input/output data and their roles are described using additional classes for the method, so that any kind of relation between input/output data can be covered.

One of the important details here is that e.g. many data sets used by one activity may have different roles for that process (one file is a parameter file, another one is the raw image, and the third one is the dark field that should be subtracted). Since these roles are very important for an activity, they have to be included in the provenance model. These roles don't have to be unique (see *[OPM]*, after Definition 10), many data sets may have the same role for a process (e.g. raw image input). The IVOA community should specify and describe the possible roles for each method (which is the template for the activity), in order to facilitate interoperability.

OPM also suggests to allow hierarchical descriptions, i.e. allow to include different ways of getting from data set A to data set B, with different levels of detail. This needs to be discussed further. (After talk at InterOp-Madrid, May 2014, some people said that this would be important for them!)

W3C describes a collection as an aggregation of entities. This is a useful concept to adapt here, since it allows to treat many data (e.g. the RAVE survey) as one entity.

TODO: more details about W3C; explain reasons, why the other relations (e.g. Communication) are not needed. Should we include specific agent-relations like attribution, delegation?

2. A Provenance Data Model

We describe here the core concepts for modelling provenance. The resulting model could then be reused as a kind of pattern everywhere where provenance is needed in the VO. [TODO: How exactly may be described here as well or even in the other data models.]

Currently, we are still debating which way to go: a model using prototypes similar to *[std:SimDM]* or rather a variant of the W3C-model. We present them here along with some of their advantages/diadvantages.

2.1. A model using prototypes

This model is inspired by *[std:SimDM]*, a data model for simulation data published in May 2013. Especially we copied the central classes: Protocol and Experiment. A Protocol is the plan or method which describes how to do something whereas Experiment is the actual execution of this plan. Defining such protocols allows them to be reused, which is very useful when performing series of experiments of the same type, as is typically done in astronomy. For our provenance mdoel, we use the more generic term "Activity" instead of Experiment, as it us used by the W3C Provenance Data Model (*[std:W3CProvDM]*). Instead of "Protocol", we use the term "Method", for not confusing it with e.g. IVOA protocols.

The class diagram generated with MagicDraw Community Edition is given in the following figure; the details are described in the next sections.

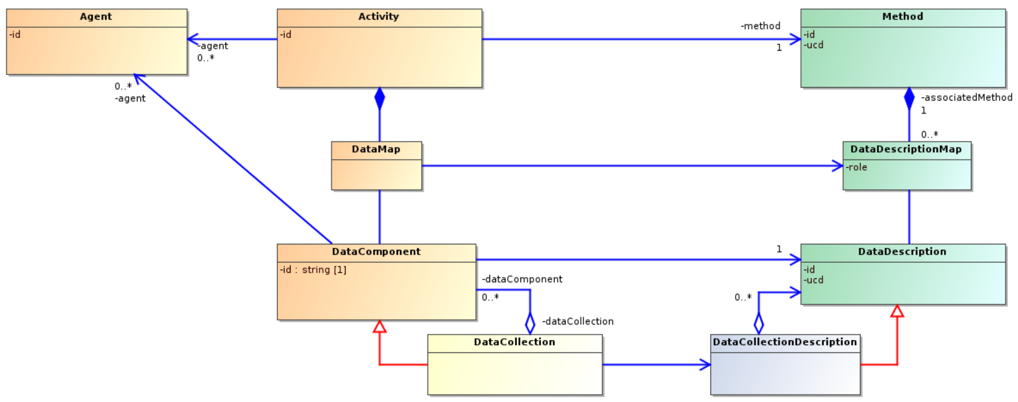

Figure 2: The (current) Provenance Data Model class diagram. It will certainly change again, so use it with care. In this version, we stripped it now from nearly all derived classes, so we can concentrate on the core elements. The green and blue boxes belong to 'prototype' classes.

2.1.1. Activity and method

Activity corresponds to Experiment and Method to Protocol from *[std:SimDM]*.

Examples for astronomical activities are observations and processing steps like flat-fielding, correcting bias, astrometric calibration etc. The type or method underlying an activity is specified by the corresponding *Method* class. This could be, for instance, the name of the code used to perform an activity or a more general description. An activity is a concrete case of using such a method, thus it has to refer to a corresponding method.

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

2.1.2. Data and DataDescription

We define a general "Data" class (actually: DataComponent, see next section), which can represent either an individual data item or a collection of data. The W3C equivalent to this class is called "Entity"; we use "Data" instead, since we do not aim to model cars and abstract ideas here, but more concretely astronomical data of any kind.

Following the scheme that each "activity" is described by the more general "method", we associate each input or output data set with a corresponding description of the data type. Each method usually makes assumptions on the structure of input data (FITS file, data table, binary file of a certain structure ...) and produces a certain form of output data, which should be described in the general "DataDescription" class, so that it can be reused for multiple instances of this class.

There has been some dispute (before and at the InterOp in Madrid, May 2014) about the different roles of input and output data. Since the results of one activity can be input data for another activity, the structure of input and output data is necessarily the same. The link from an "Activity" to the corresponding "Data" object could tell then if the piece of data is used as input or output (see figure at 2.1.6).

However, input data can take different roles in an activity (auxiliary data like a parameter file, two images, with one that needs to be subtracted from the other) and it must be made explicit which data component that is used by or generated from an activity fulfills which role. In W3C, this is partially solved by adding a derivation relation between the entities (data). Here, we add a mapping-class between activity and data as well as between method and dataDescription. The mapping-class at the description side, i.e. between the Method and its DataComponentDescriptions, contains additionally a role for each relation, e.g. parameter, dark frame, raw image, etc. If a dataComponent is used as input to an activity or results from it, will become clear with these roles.

Note: Suggestion by Jochen: What about defining a Role-class for methods? This way each method should define the roles that are allowed for the method (probably taken from a specified list of vocabularies), enhancing inter-operability.

Without the mapping tables, the relation between activity/method and data/datadescription would be an aggregation relation, or in other words: an association with the aggregationKind "shared". That would be required to ensure that all DataComponents linked to an activity (either as input or output) will survive if the activity is destroyed, since they are almost always shared with other activities.

By using the mapping tables we make the role of a dataComponent in an activity more explicit and thus can replace the aggregation by a composition relation to the activity/method and simple associations to the individual data components and their descriptions.

2.1.3. Data Collections - Composite pattern

There are two major data classes:

- data collections
  e.g. RAVE-DR4 with its databases and database tables, SDSS DR9 with all its tables and files, all files from one observing slot
- individual data components
  e.g. a file, table in a database, parameter value, an image, a fits-file containing a table and image

Data can only be grouped to a data collection, if they have the same origin (i.e. they were produced by the same activity, i.e. they have the same provenance). We would leave it to the person/tool recording provenance to decide, how detailed a data set will be separated into individual items. It is also possible to just record provenance for e.g. RAVE DR4 as a whole, without listing everything that belongs to DR4. The level of detail could then be specified via the DataDescription.

For our data model, it would be desirable to be able to treat data collections and individual data items in the same way, with the same interface. This demand led us to the **Composite design pattern**:

We introduce an abstract class called "DataComponent", which includes the basic properties and functionality for both, data items like files or parameter values or a complete data set. Such common properties are e.g. the link to the activity which created the data/dataCollection, a creation date/time and a link to a storage-object (which is not further specified in this data model). Each DataComponent can either be a single item (like a file, database table or a parameter) or a DataSet. A DataCollection contains additionally a (non-empty) list of child-DataComponents, which could again contain a list of further children etc. This way, one could represent complete data trees, if necessary. Each DataComponent also has a link to its parent, which would be emtpy (or point to itself) for the root-DataComponent. We hope that such a representation would make it easier for clients to handle data objects, since they don't have to make the distinction between data items and data collections.

2.1.4. Agent

In SimDM, someone performing a certain experiment is called "Contact", the W3C provenance data model suggests the term "Agent", so we adopted it here. We want to describe someone who is responsible for an activity, e.g. who pressed a button, ran a script or performed the observation. The agent could be a single person (specified by name), a group of persons (e.g. MUSE WISE Team), project/organization (RAVE) or an institute. For each of them a name should be specified.

It is desired to have an agent given for each activity, but it is not enforced (hence 0..*). It would also increase the value of the given information if the (current) affiliation of the agent and a project leader/group leader were specified in order to maximize the chance of finding someone later on. The agent should not only be used for getting contact information, but also to fulfill our "Attribution" requirement, so that proper credit goes to the right people/projects. To this end, it could also make sense to add a relation between a DataComponent and Agent, similar to W3C's wasAttributedTo-relation.

TODO: Specify pre-defined contact or agent roles to choose from, in order maximize interoperability (see SimDM: owner, creator, publisher, contributer; any more?)

2.1.5. Storage

The modeling of storage is not included in this model. It would be useful to be able to make a reference from the DataComponent to a storage object that contains the link to where the data is stored. But we haven't fixed the details here.

2.1.6. Links Between Data

It would be convenient if each data object or even each file ("Storage" object) gets a unique "dataId" that can be referenced. If several copies have been made out of a data set and one of them is corrupted, it would even be useful to know exactly which copy was used by a given activity.

Maybe DataLink can help here?

We need to be able to refer from an activity to the ResultData of another activity. A possible data flow is shown in the figure below:

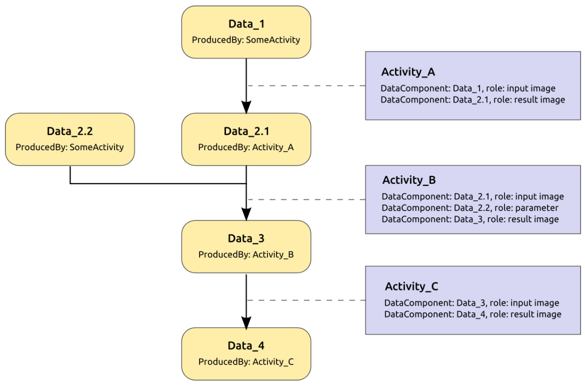
Data_1
ProducedBy: SomeActivity
Activity_A
DataComponent: Data_1, role: input image
DataComponent: Data_2.1, role: result image
Data_2.2
ProducedBy: SomeActivity
Data_2.1
ProducedBy: Activity_A
Activity_B
DataComponent: Data_2.1, role: input image
DataComponent: Data_2.2, role: parameter
DataComponent: Data_3, role: result image
Data_3
ProducedBy: Activity_B
Activity_C
DataComponent: Data_3, role: input image
DataComponent: Data_4, role: result image
Data_4
ProducedBy: Activity_C

Figure 3: The black arrows indicate the data flow, gray lines just show which activity would be attached to the step. It is defined by the role in the corresponding Activity whether data is used as input or output.

2.1.7. Calibration data

The calibration data set consists of images that can be used to calibrate the raw data. It is not necessary to mention them explicitly in the model, they are just another dataSet that is used by activities with a calibration-method.

2.1.8. Ambient Conditions

Ambient conditions are environmental properties, which are special in a way: they represent the part of an activity, usually an observation, which cannot be (fully) controlled by an observer, in contrast to other data that can be fully reproduced. Nevertheless we decided that they can be fully described by our data class already and don't need a separate class in our data model. Our model can then also take into account that a certain observation method requires special ambient conditions, already defined via the method (e.g. radio observations rely on different ambient conditions than observations of gamma rays), just following our data - data description scheme.

Ambient conditions are recorded for a certain time (executionDate) and are usually only valid for a certain time interval. This time interval should be recorded with a validity-attribute for DataComponents.

We wondered if ambient conditions could also play a role for some processing steps or any other activity besides observations? Is there any additional step performed in which room temperature etc. may play a role and thus should be recorded? The only example that came to our minds was the storage of photographic plates, where the humidity and temperature variations can affect the quality of the photographic plates.

2.1.9. Instrument Characteristics

In contrast to ambient conditions, instrument characteristics do (usually) not change from one observation to the other, so they are static, strictly related to the instrument. All the characteristics could be described either as key-value pairs directly with the observation or just as data, using the DataComponent class. One would then link the instrument characteristics as a type of input data set to a certain observation activity. Thus we don't need a separate Instrument or Device class.

One should also keep in mind that some instrument related parameters can change within time, e.g. the CCD temperature. The instruments can also change within time because of aging.

2.1.10. Quality

We could simply define additional attributes for each "Activity" or "Data" object, i.e. zero, one, or more properties in the form of key-value pairs. We could use a "Quality" namespace to mark a keyword as quality-related:

- quality.comment: [some text]
- quality.seeing: [some value]

The values could range from a float number to free text.

2.1.11. Discussion

This model was established with having a database implementation in mind. However, we'll look again at W3C in the next section, which may offer simpler possibilities to store provenance with the dataComponents themselves, e.g. as an additional structure in fits-headers.

A model using prototypes has the advantage of normalisation: methods could be described once and for all at some place (this *some place* is actually the crucial point here!) and then be reused when describing the actual provenance of certain dataComponents and activities. However, building such a look-up place for all the possible methods and dataDescriptions is a quite challenging task -- it will probably never be complete. There's also the issue of persistent identifiers/broken links to consider. Normalisation is useful for closed systems, e.g. for describing the provenance for data produced by a certain pipeline (e.g. MuseWise system) or with workflow tools or when a task needs to be repeated many times. However, the VO is quite the contrary of a closed system and we need to keep an eye on what is actually achievable.

2.2. A simpler model based on W3C

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

Thus, we consider the W3C-model again, and adjust it by replacing the used- and wasGeneratedFrom-relations by a mapping class between activity and entity in the following class diagram:

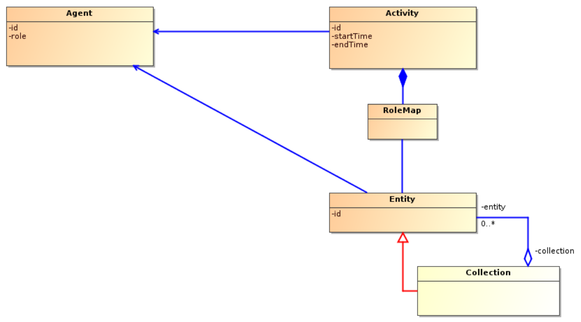

Figure 4: An adjusted W3C model for provenance of astronomical data.

The different methods and data types are here hidden in pre-defined vocabulary lists. An example for writing down the provenance for a stacked image in W3C and with the protoype-model is given in the accompanying files at `hprov-example-incl-prototypes.txt` and `prov-example-w3c.txt` in https://code.google.com/p/volute/source/browse/trunk/projects/dm/provenance/description/.

2.2.1. Discussion

Expressing provenance for a stacked image with this smaller set of classes may be simpler, but on the other hand constructing a database schema becomes much harder. The question is: could we just derive a simple exchange format from the prototype-model? Should we be instance-driven here? Which model would be more useful?

References

[std:W3CProvDM] Khalid Belhajjame, Reza B'Far, James Cheney, Sam Coppens, Stephen Cresswell, Yolanda Gil, Paul Groth, Graham Klyne, Timothy Lebo, Jim McCusker, Simon Miles, James Myers, Satya Sahoo, and Curt Tilmes.
PROV-DM: The prov data model. W3C Recommendation, April 2013.

[std:RFC2119] Scott Bradner.
Key words for use in RFCs to indicate requirement levels. RFC 2119, March 1997.

[std:SimDM] Gerard Lemson, Laurent Bourgès, Miguel Cerviño, Claudio Gheller, Norman Gray, Franck LePetit, Mireille Louys, Benjamin Ooghe, Rick Wagner, and Hervé Wozniak.
Simulation data model, version 1.0. IVOA Recommendation, May 2012.