International
Virtual
Observatory
Alliance

VO-DML Tools

Version 1.0

IVOA Note 2025-11-11

Working Group
Data Models
This version
https://www.ivoa.net/documents/Notes/VodmlTools/20251111
Latest version
https://www.ivoa.net/documents/Notes/VodmlTools
Previous versions
This is the first public release
Author(s)
Paul Harrison
Editor(s)

Version Control
Revision 06faf6d-dirty, 2025-11-11 14:35:04 +0000

Abstract

The IVOA standard modelling language VO-DML Lemson and Laurino et al. (2018) was developed with an associated set of tooling that could be used in the creation of data models expressed in VO-DML. The VO-DML standard does not go into any detail about the tooling as it describes only the language itself. This note attempts to fill that gap by describing the development of the tooling, and give an outline of its functionality. In addition this document promotes the idea that the VO-DML tools are the de-facto standard way to create the schemas for serializations of model instances.

Status of this document

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

A list of current IVOA Recommendations and other technical documents can be found in the IVOA document repository[1].

Contents

Acknowledgments

Although the author is the current lead maintainer of the VO-DML Tools, the tools are based on the original contributions principally of Gerard Lemson and Laurent Bourgès, and presumably associated contributions from others working on SIM-DM (Lemson and Wozniak et al., 2012) and the earlier VO-URP[2] project.

[1] https://www.ivoa.net/documents/

[2] https://github.com/glemson/vo-urp

1 Introduction

VO-DML Lemson and Laurino et al. (2018) was developed as a machine-readable language for creating data models. This was done to promote re-use and rigour in the model definitions, in contrast to the existing practice of defining models only by description in documentation, which allowed an unconstrained set of possibilities, although it was common at least to use UML *UML — Unified Modeling Language* (n.d.). There was a desire to continue to use UML tools to create data models which meant that there was a requirement for some tooling to convert the UML representation to VO-DML - this requirement was met by the development of the VO-DML tools. Although the VO-DML tools were not described in the VO-DML standard as it is possible to create VO-DML definitions 'by hand' in a text editor, this is increasingly infeasible to do as the models grow in complexity - at a minimum there needs to be a mechanism to ensure that any models created do not contain errors and are valid against the VO-DML design criteria. One of the earliest functionalities of the VO-DML tools was to produce standard documentation and diagrams of the data models, which again meant that, in practical terms, the use of the VO-DML tools was a requirement when developing a data model, as the production of such documentation was a requirement of the VO-DML standard.

In addition, one of the main purposes of creating a machine-readable definition of a data model is to be able to then generate products from this representation. However, it is at this point that the VO-DML standard does not make any normative requirements - there is some discussion of how this might be done in in Appendix B of various serialization formats, but not described in sufficient detail to produce interoperable formats. One of the main aims of this document is to propose that the implementation of these serializations that has been done in the VO-DML tools is the de-facto standard for how such concrete serializations are created from VO-DML models.

2 A Brief History of VO-DML Tools

This section outlines the author's recollection of the timeline of the development of the VO-DML tools. It is possible, of course, to check the veracity of these recollections in the version control change logs for VO-URP[3] and the VO-DML Tools[4]

As mentioned in the acknowledgments, the VO-DML tools have their origins in the Simulation Data Model and VO-URP projects, which included

[3] https://github.com/glemson/vo-urp/commits/master/
[4] https://github.com/ivoa/vo-dml/commits/master/

Java code generation from a precursor meta-language that was similar to VO-DML. In the process of standardizing the VO-DML meta-language, some capabilities of the precursor language were dropped, as well the ability of the tools to create Java code that would serialize instances of the model described. The capabilities that the tools possessed at the time that VO-DML 1.0 was designated a recommendation included;

- Being able to transform the output of several UML modelling tools.
- Detailed validation of the VO-DML model against the standard.
- Creating model documentation as a single HTML page.

Starting in early 2021 the author had a requirement to design some new data models and create services against them.[5] It seemed sensible to try to use VO-DML to define the data models and try to revive the code generation facilities of VO-URP. The way that the tooling was distributed was also changed see section 2. The facilities that the current tooling provides are

- Creating an XML schema that validates serialized instances of a particular data model
- Creating a JSON schema that similarly can be used to validate serialized instances
- Creating a TAPSchema representation (an XML serialization of the TAPSchemaDM also defined in VO-DML[6])
- Generation of Java code that can create instances of the data model and implements the serializations listed above
- Generation of Python code that similarly does the same[7]
- Generation of multi page documentation for the data model, which has several advantages over the single page HTML documentation;
  - has better navigation.
  - resizes itself for modern mobile screen sizes.
  - has a localised diagram for each VO-DML element that shows its connection to the rest of the model.
- Being able to author a VO-DML model in VODSL[8] as well as UML.

[5] This is the Polaris proposal tool https://github.com/orppst which uses ProposalDM https://ivoa.github.io/ProposalDM/

[6] https://ivoa.github.io/TAPSchemaDM/

[7] the Python code generation is not as sophisticated as the Java code generation

[8] https://www.ivoa.net/documents/Notes/VODSL/

2.1 Technologies used to write the Tools

As the VO-DML is expressed in XML the VO-DML tools made the early choice to implement all of the business logic in XSLT. At the time of publishing VO-VML 1.0 the running of the various XSLT transformations was orchestrated using the Ant[9] build tool. The more recent versions of the VO-DML tools have been rewritten so that the Gradle[10] build tool, which has the following advantages of the previous Ant orchestration;

- all of the associated XSLT and other files that are needed for the tools functionality can be packaged as a gradle plug-in - this allows use of the tools by a simple textual reference to the plugin, rather than having to copy the required files from the VO-DML repository.
- gradle makes it easy to include other dependencies via maven repositories

The choice of orchestration tools does look rather 'Java-centric' and it reflects the fact that the initial target of the code generation target was Java. However, this does not mean that the tools are exclusively of utility to Java programmers - Indeed if only interested in the generated schema and documentation there is no particular language association (apart from a requirement of an installed Java Virtual Machine to run the tools). The tools can equally generate python code, and there is a prototype orchestration that is implemented in Python - however, this orchestration implementation is not as comprehensive in its functionality as the Gradle orchestration.

3 Using the VO-DML tools

This document is deliberately deficient in detail on exactly how to use the VO-DML tools. This has been done because the intention is that this detail of the use of the tools will be maintained as on-line documentation at https://ivoa.github.io/vo-dml/ rather than published via the usual IVOA document publishing route.

4 Serialization

One of the principle advantages of having a machine readable version of a data model is that it is possible to further transform the model into other machine readable representations. One of the principle reasons to create a data model in the first place is to be able to exchange instances of that data model between different systems that might be interested in the content of

[9] https://ant.apache.org
[10] https://gradle.org

those instances and be sure that the same interpretation will be made of the instance in each circumstance. Interchange is done by a process known as serialization where the model instances are transformed into a form that is easy to transport via some transport mechanism. This often means that the model instances are transformed into a textual form for transmission. In recent times two popular methods of doing this have been XML and JSON - these formats have their own meta-models that are typically generic enough to be able to express instances of complex data models. However, because they are generic they typically cannot express the same semanic information that is in a data model. In addition these formats, because they were often defined for interchange purposes will have their own schema systems to be able to validate instances against their meta-model. It is therefore useful to be able to generate schema from a VO-DML model so that the XML or JSON tools might be able to validate serialization instances of the data model, as they cannot directly understand the VO-DML.

4.1 References in serialization

4.2 Schema representations

Depending to the complexity of a particular schema language and serializationformat, there might be several different styles that can be used to represent data model instance serializations - A possibility that occurs in XML is whether a particular VO-DML attribute should be represented by an XML element or an XML attribute - somtimes, for instance, if the VO-DML is a DataType then only an XML element would be the only formulation capable of being able to represent the structure in a DataType - therefore the choice between XML element and XML attribute cannot be a global choice, but can only be made on an attribute by attribute basis. A purely stylistic choice that can be made with XML for instance is where an XML element in a sequence that has a multiplicity of greater than one should be 'wrapped' by another element, and although this does not alter any of the semantics of the data model being represented it often 'reads better' to the human eye when looking at such an XML structure.

5 Standardization

6 Future Directions

It should be noted that the VO-DML tools are in a state of active development and have not reached what the author would consider a 1.0 status. The development is controlled by using the usual management mechanisms available in GitHub - in particular issues are created for suggested enhancements and bug fixes and code is controlled via pull requests.

The development of the VO-DML Tools also is a test-bench for future enhancements to the VO-DML language and standard, as new developments will always be prototyped in the tools before being suggested for the the standardization track.

A Changes from Previous Versions

No previous versions yet.

References

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. doi:10.5479/ADS/bib/2018ivoa.spec.0910L, https://ui.adsabs.harvard.edu/abs/2018ivoa.spec.0910L.

Lemson, G., Wozniak, H., Bourges, L., Cervino, M., Gheller, C., Gray, N., LePetit, F., Louys, M., Ooghe, B. and Wagner, R. (2012), 'Simulation Data Model Version 1.0', IVOA Recommendation 03 May 2012, arXiv:1402.4744. doi:10.5479/ADS/bib/2012ivoa.spec.0503L, https://ui.adsabs.harvard.edu/abs/2012ivoa.spec.0503L.

UML — Unified Modeling Language (n.d.). https://www.omg.org/spec/UML/.