International
Virtual
Observatory
Alliance

Mapping Data Models to VOTable
Version 1.0-20170323

IVOA Working Draft 2017-03-23

Working group

This version
http://www.ivoa.net/documents/MappingDMtoVOTable/20170323
Latest version
http://www.ivoa.net/documents/MappingDMtoVOTable
Previous versions

Author(s)
Gerard Lemson, Omar Laurino, Tom Donaldson, Mark Cresitello-Dittmar, Laurent Michel, Patrick Dowler, Markus Demleitner, Matthew Graham, Jesus Salgado
Editor(s)
Omar Laurino, Gerard Lemson

Abstract

Data providers and curators provide a great deal of metadata with their data files: this metadata is invaluable for users and for Virtual Observatory software developers. In order to be interoperable, the metadata must refer to common Data Models. This specification defines a scheme for annotating VOTable instances in a standard, consistent, interoperable fashion, so that each piece of metadata can unambiguously refer to the correct Data Model element it expresses, assuming there is a suitable data model. With this specification, data providers can unambiguously and completely represent Data Model instances in the VOTable format, and clients can build *faithful* representations of such instances. The mapping is operated through opaque, portable strings.

Status of This Document

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

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

Contents

Listings

History of this document

TODO migrate document’s history

1 Introduction

Data providers put a lot of effort in organizing and maintaining metadata that precisely describes their data files. This information is invaluable for users and for software developers that provide users with user-friendly VO-enabled applications. For example, such metadata can characterize the different axes of the reference system in which the data is expressed, or the history of a measurement, like the publication where the measurement was drawn from, the calibration type, and so forth. In order to be interoperable, this metadata must refer to some Data Model that is known to all parties: the IVOA defines and maintains such standardized Data Models that describe astronomical data in an abstract, interoperable way.

In order to enable such interoperable, extensible, portable annotation of data files, one needs:

- A language to unambiguously and efficiently describe Data Models and their elements’ identifiers (VO-DML, (Lemson 2015)).
- Pointers linking a specific piece of information (data or metadata) to the Data Model element it represents[1].
- A mapping specification that unambiguously describes the mapping strategies that lead to faithful representations of Data Model instances in a specific format.

[1]This used to be the assumed role of the `@utype` attribute in VOTable and for example TAP. This document is based on the new `VODML` element in `VOTable` 1.4 (Ochsenbein 1900).

Without a consistent language for describing Data Models there can be no interoperability among them, through reuse of models by models, or in their use in other specifications. Such a language must be expressive and formal enough to enable the serialization of data types of growing complexity and the development of reusable, extensible software components and libraries that can make the technological uptake of the VO standards seamless and scalable.

For serializations to non-standard representations one needs to map the abstract Data Model to a particular format meta-model. For instance, the VOTable format defines `RESOURCE`s, `TABLE`s, `PARAM`s, `FIELD`s, and so forth, and provides explicit attributes such as `units`, `UCD`s, and `utypes`: in order to represent instances of a Data Model, one needs to define an unambiguous mapping between these meta-model elements and the Data Model language, so to make it possible for software to be able to parse a file according to its Data Model and to Data Providers to mark up their data products.

While one might argue that a standard for portable, interoperable Data Model representation would have been required before one could think about such a mapping, we are specifying it only at a later stage. In particular several different interpretations of `UTYPE`s have been proposed and used (Graham 2013). This specification aims to resolve this ambiguity.

As a matter of fact, existing files and services can be made compliant according to this specification by simply *adding* annotations and keeping the old ones. So they do not need to *change* them in such a way that would necessarily make them incompatible with existing software.

Several sections of this document are utterly informative: in particular, the appendices provide more information about the impact of this specification to the current and future IVOA practices.

This specification describes how to represent Data Model instances using the VOTable schema. This representation uses the `<VODML>` element introduced for this purpose in VOTable v1.4 (Ochsenbein 1900) and the structure of the VOTable meta-model elements to indicate how instances of data models are stored in VOTable documents. We show many examples and give a complete listing of allowed mapping patterns.

In sections 2 to 5 we give an introduction to why and how the VODML elements can be used to hold pointers into the data models.

Section 6 is a rigorous listing of all valid annotations, and the normative part of the specification.

The appendices contain additional material. Section 7 describes the VODML annotation element that was added to the VOTable schema to support this mapping specification. Section 2.4 describes different types of client software and how they could deal with VOTables annotated according to the current specification.

Throughout the document we will refer to some real or example Data Models. Please remember that such models have been designed to be fairly simple, yet complex enough to illustrate all the possible constructs that this specification covers. They are not to be intended as actual DMs, nor, by any means, this specification suggests their adoption by the IVOA or by users and or Data Providers. In some cases we refer to actual DMs in order to provide an idea of how this specification relates to real life cases involving actual DMs.

2 Use Cases

2.1 General Remarks

This specification provides a standardized mechanism for annotating VOTables according to data models. Thus, it enables:

- Data providers to annotate VOTables so to faithfully map VOTable contents to one or more data models, as long as such Data Models are expressed according to the VO-DML standard (Lemson 2015). In other terms, they can *serialize* data model *instances* in a standard, interoperable way. Some examples are provided below as concrete use cases.
- Service clients to faithfully reconstruct the semantics of the data model instances they consume in VOTable format. Some concrete examples are also provided below.

One of the main goals of this specification is also to alleviate data modelers from the burden of defining special serialization strategies for their data models, at least in most cases. Specialized serializations might be defined if there are special constraints in term of efficiency or effectiveness which require specialized serialization schemes.

As a corollary to the above paragraph, client applications can also be implemented on top of standardized Input/Output libraries that implement the present specification, as the serialization mechanisms are standardized across data models. Without this specification clients would need to be coded against specialized serializations for each data model. Similarly, data providers can now serialize instances of any data model to VOTable using the same annotation mechanism.

As a result, this mapping specification should enable a large number of concrete use cases to be implemented, by reducing the annotation burden on both data providers and data consumers, improving the overall interoperability, at least for what VOTable is concerned.

This document also represents a template for mapping data model instances to other formats (see sec. 2.5).

2.2 Concrete Use Cases

2.2.1 STC clients

A typical usage scenario may be a VOTable client that is sensitive to certain models only, say Space Time Coordinates (STC). Such a tool may be written to understand annotations for instance of STC types, manipulate such instances, and write them back to disk.

By finding an element mapped to a type definition from the STC model, the application may infer for example that it represents a coordinate on the sky and use this information according to its requirements. For example, the client may convert all positions expressed in a certain coordinate frame to a different coordinate frame through some specific transform.

Note that the client may never parse any VO-DML description files, as the knowledge about the data model may be assumed when the client code is developed.

Also, the STC annotations are the same in all the contexts in which an STC type is used. So, for instance, if a VOTable describes catalogues of sources, and each source has a position attribute describing its coordinates in a specific reference frame, the *instance* describing the source's position would be annotated in the exact same way as, say, the central position of a cone search query.

So, the STC tool may not necessarily understand other models where STC types are *used*, but it may still be able to find the instances of those types.

Note that the same file may contain multiple tables and in any case STC model instances defined in multiple coordinate systems or frames.

2.2.2 VO-enabled plotting and fitting applications

An application whose main requirement is to display, plot, and/or fit data cannot be required to be aware of *all* astronomical data models. However, if these data models shared some common representation of quantities, their errors, and their units, the application may discover these pieces of information and structure a plot, or perform a fit, with minimal user input: each point may be associated with an error bar, upper/lower limits, and other metadata. The application remains mostly model-agnostic: it is not required to *understand* high-level concepts like Spectrum, or Photometry.

Also, knowledge about the basic building-block types like quantities and coordinates, may be hard-coded during development.

2.2.3 Data discovery portals

Data discovery portals allow users to query VO services, display metadata, filter responses, and fetch datasets.

While these applications may not be particularly interested in specific data models, standardized annotation mechanisms may allow their developers to dynamically capture the structure of the metadata, and provide better exploratory tools rather than flat tables.

Consider for example filtering tables according to an arbitrary physical quantity, say for instance the spectral coverage of a spectrum, the filter with which an image was taken, or the luminosity of a source in a certain band. The portal may provide a friendly interface that allows users to select the physical quantities using standardized representations. It may do so dynamically for all the pieces of metadata present in the dataset, rather than limiting this functionality to a set of hard-coded metadata properties.

Also, the application may group data and metadata in a tree of concepts the user is familiar with, rather than flattening the instrinsic structure of the data.

By allowing such applications to faithfully represent data model instances the way they were curated and annotated by data providers, and according to the scientific domain with which users are familiar, this specification may enable users to easily make sense of the file contents, even if the application lacks any knowledge of high-level data models.

Parsing VO-DML description files may be useful to provide the user with even more, and more accurate, information.

2.2.4 Color-color diagrams

The creation of a color-color diagram requires knowledge of the semantics of the rows and columns in a table, e.g. a source catalog. Also, some columns may be grouped together in a structured way, e.g. a luminosity measurement that goes with its error and a reference to the photometry filter it is associated with, although such columns might not be adjacent in the file.

Usually, plotting applications are not aware of the semantics of the columns in a table, and users may have to select all the relevant columns in order to produce a meaningful plot. They may also have to convert between units, and so on.

With a standardized annotation and knowledge of the annotations for basic quantities a plotting application may find that there are luminosity measurements in a VOTable and allow users to display color-color diagrams, with error bars, and possibly with domain-specific actions like unit conversions, seamlessly and requiring minimal input.

There are many examples of very specific use cases that may be implemented by science applications that are aware of the semantics of specific models and their annotations.

2.2.5 Validators

The existence of an explicit data model representation language and of a precise, unambiguous mapping specification enables the creation of universal validators, just as it happens for XML and XSD: the validator may parse the data model descriptions imported by the VOTable and check that the file represents valid instances of one or more data models.

2.2.6 VO Publishing Helper

There is some complexity involved in understanding how to publish data in the Virtual Observatory. The availability of a standard for serializing data model instances can provide tools for publishers to build templates of their responses.

Such application may help data providers in interactively mapping Data Models elements to their files or DB tables, either producing a VOTable template with the appropriate annotations, or by creating a DAL service on the fly.

The application is not required to be model-aware, since it may get all the information from the standardized description files and the mapping specification.

2.2.7 VO Importer

Users and Data Providers may have non-compliant files that they want to convert to a VO-compliant format according to some data models: a generic, model-unaware importer application may allow them to do so for any standard data model with a proper description file.

2.3 Generic Use Cases

This section generalizes the previous one by stating the same use cases in a more abstract, generic formulation. It also adds some more generic use cases that this specification addresses or enables.

2.3.1 Serialize and de-serialize instances according to a data model

Data providers may want to serialize data and metadata in VOTable according to a specific data model with a standardized description.

A client may build an in-memory faithful representation of that instance according to the data provider's annotations, assuming the knowledge of a finite set of data model identifiers, of a full data model, or by parsing standard VO-DML data model specifications.

2.3.2 Annotate files according to multiple models

Data providers may find it useful to annotate a file according to different data models for different classes of data consumers. Also, they may decide to provide annotations according to different versions of a specific data model, to favor backward compatibility with older clients.

2.3.3 Representing cross matches and linking files together

It is often the case that two or more files or tables represent different pieces of information regarding the same astronomical sources of objects. In these cases, one or more columns usually are used as keys to identify instances in such tables.

For instance, the output of a cross-matching service may provide the IDs of the cross matched sources along with some data regarding the cross-matching process, while most of the data about the sources themselves may be stored in different tables.

This is a very common relational pattern. A standardized annotation with Object-Relational Mapping capabilities may be used to connect different tables and provide users with a unified view according to data models the user is familiar with.

It is then possible to link different views of the datasets, with an additional layer of semantics. For instance, an application may display the image of a region of the sky, and a catalogue may contain information about sources in the catalogue.

A user may want to link the image to the catalogue. When no a-priori link between the image coordinates and the positions in the catalogue is known, users need to set such links themselves, by selecting the relevant columns. With standardized annotations according to specific data models applications can figure out the links themselves, and ask the user to intervene only when there is ambiguity or lack of information, or when the user wants to make a custom link.

Similarly, one may produce a color-color diagram from magnitudes stored in different tables as long as there is a known mapping from source identifiers among tables, and a standardized annotation of all the tables involved.

2.3.4 Mission-specific data model extensions

One may identify data and metadata features that are common to a certain astronomical domain, e.g. catalogues of astronomical sources. These are the models the IVOA sanctions in standards.

However, different missions, or archives, or applications will most certainly have specific additional features that are not captured by such common, standardized data models.

One may express such extensions and their instances in such a way that: * data providers may easily annotate specialized instances, including the additional information, in a standardized, interoperable fashion. * clients of the common, standard data models may still find instances of these parent models in files serializing specialized instanced. * a model annotation that is valid according to a specialized data model is also valid according to the parent model.

This use case can be formalized in terms of inheritance and polymorphism.

Inheritance allows models to specialize types defined in other data models. Polymorphism is the common object-oriented design concept that says that the value of a property may be an instance of a *subtype* of the declared type.

Typed languages such as Java support a casting operation, which provides more information to the interpreter about the type it may expect a certain instance to be.

A client must be able to identify the information about a standard type, even if the serialization includes instances of its subtypes. Similarly, a client should have enough information to *cast* an instance from the declared type to the actual subtype.

2.4 Growing complexity: simple, advanced, and guru clients

According to the use cases depicted above, we can classify clients in terms of how they parse the VOTable in order to harvest its content. Of course, in the real word such distinction is somewhat fuzzy, but this section tries and describe the different levels of usage of this specification.

This classification is useful because it shows how implementations can be based on different assumptions. Some clients can focus on few hard-coded elements, while other clients can dynamically address complex tasks.

2.4.1 Simple clients

We say that a client is *simple* if:

- it does not parse the VO-DML description file

- it assumes the a priori knowledge of one or more data models
- it discovers information by looking for a set of predefined vodml-refs in the VOTable

In other terms, a simple client has knowledge of the data model it is sensitive to, and simply discovers information useful to its own use cases by traversing the `VODML` element.

Examples of such clients are the DAL service clients that allow users to discover and fetch datasets. They may just inspect the response of a service and present the user with a subset of its metadata. They do not *reason* on the content, and they are not interested in the structure of the serialized objects.

If such clients allow users to download the files that they load into memory, they should make sure to preserve the structure of the metadata, so to be interoperable with other applications that might ingest the same file at a later stage.

2.4.2 Advanced clients

We say that a client is *advanced* if:

- it does not parse the VO-DML description file
- it is interested in the structure of the serialized instances
- can parse the elements defined in this specification

Examples of such clients are discovery portals or science applications that display information to the user in a structured way, e.g. by plotting it, or by displaying its metadata in a user-friendly format. Possibly, such applications may allow users to save versions of the serialization after it has been manipulated.

Such clients may not assume any knowledge of any specific data models. In some cases they may assume knowledge of some types from some basic, common data models, to perform additional tasks.

Even if such applications may be model-unaware, they may allow users to build Boolean filters on a table, using a user-friendly tree representing the whole metadata. This exposes all the metadata provided by the Data Provider in a way that may not be meaningful for the application, but that may be meaningful for the user.

2.4.3 Guru clients

We say that a client falls into this category if:

- it parses the VO-DML descriptions
- it does not assume any a priori knowledge of any data models.

Such applications can, for example, dynamically allow users and data providers to map their files or databases to the IVOA data models in order to make them compliant, or display the content of any file annotated according to this standard.

This specification allows the creation of universal validators equivalent to the XML/XSD ones.

It also allows the creation of VO-enabled frameworks and reusable libraries. For instance, a Python universal I/O library can parse any VOTable according to the data models it uses, and dynamically build structured objects on the fly, so that users can directly interact with those objects or use them in their scripts or in science applications, and then save the results in a VO-compliant format.

Java and Python guru clients could automatically generate interfaces for representing data models and dynamically implement those interfaces at runtime, maybe building different views of the same file in different contexts.

Notice that guru frameworks and libraries can be used to build advanced or even simple applications in a user-friendly way, abstracting the developers from the technical details of the standards and using scientific concepts as first class citizens instead.

2.5 Formats other than VOTable

We want to explicitly note that this specification covers the VOTable format only.

Other mapping specifications can and will provide standardized strategies for mapping Data Models to formats other than VOTable.

Part of the implementation efforts related to the present specification was to validate the standard against prototype serializations in JSON and YAML formats.

Mapping specifications targeting additional formats can use this document as a template.

3 The need for a mapping language

When encountering a data container, i.e. a file or database containing data, one may wish to interpret its contents according to some external, predefined data model. That is, one may want to try to identify and extract instances of the data model from amongst the information. For example in the *global as view* approach to information integration, one identifies elements (e.g. tables) defined in a global schema with views defined on the distributed databases[2].

If one is told that the data container is structured according to some standard serialization format of the data model, one is done. I.e. if the local database is an exact *implementation* of the global schema, one needs no special annotation mechanism to identify these instances. An example of this is an XML document conforming to an XML schema that is an exact physical *representation* of the data model. We call such representations *faithful*.

But in an information integration project like the IVOA, which aims to homogenize access to many distributed heterogeneous data sets, databases and documents are in general *not* structured according to a standard representation of some predefined, global data model. The best one may hope for is to obtain an *interpretation* of the data set, defining it as a *custom serialization* of the result of a *transformation* of the global data model[3]. For example, even if databases themselves are exact replications of a global data model, results of general queries will be such custom serializations.

To interpret such a custom serialization one generally needs extra information that can provide a *mapping* of the serialization to the original model. If the serialization *format* is known, this mapping may be given in phrases containing elements both from the serialization format and the data model. For example if our serialization contains data stored in 'rows' in one or more 'tables' that each have a unique 'name' and contain 'columns' also with a 'name', you might be able to say things like:

- The rows in this table named SOURCE contain *instances* of *object type* 'Source' as defined in *data model* 'SourceDM'.
- The *type*'s 'name' *attribute* (having *datatype* 'string', a *primitive type*) also acts as the *identifier* of the Source *instances* and is stored in the single column with ID 'name'.
- The *type's* 'classification' *attribute* is stored in the table column CLASSIFICATION (from the *data model* we know its *datatype* is an *enumeration* with certain *values*, e.g. 'star', 'galaxy', 'agn').
- The *type's* 'position' *attribute* (being of *structured data type* 'SkyCoordinate' defined in *model* 'SourceDM') is stored over the two columns RA and DEC, where RA stores the SkyCoordinate's *attribute* 'longitude', DEC stores the 'latitude' *attribute*. Both must be interpreted

[2]See, for example, http://logic.stanford.edu/dataintegration/chapters/chap01.html
[3]Or alternatively as a transformation of a (standard) serialization of the data model.

using an *instance* of the SkyCoordinateSystem *type*, This *instance* is stored in 1) another document elsewhere, referenced by a *reference* to a URI, or 2) in this document, by means of an *identifier.*

- *Instances* from the *collection* of luminosities of the Source *instances* are stored in the same row as the source itself. Columns MAG_U and ERR_U give the 'magnitude' and 'error' *attributes* of *type* LuminosityMeasurement in the "u band", an *instance* of the Filter *type.* (stored elsewhere in this document ('a *reference* to this Filter instance is …'). Columns MAG_G and ERR_G … etc.
- Luminosity *instances* also have a filter *relation* that points to instances of the PhotometryFilter *structured data type*, defined in the IVOA PhotDM model, whose *package* is imported by the SourceDM.

In this example the *emphasized* words refer to concepts defined in VO-DML, a meta-model that is used as a formal language for expressing data models. The use of such a modeling language lies in the fact that it provides formal, simple, and implementation neutral definitions of the possible structure, the 'type' and 'role' of the elements from the actual data models that one may encounter in the serialization (SourceDM). This can be used to constrain or validate the serialization, but more importantly it allows us to formulate mapping rules between the serialization format (itself a kind of meta-model) and the meta-model, independent of the particular data models used; for example rules like:

- An *object type* MUST be stored in an 'INSTANCE'.
- A '*primitive type*' MUST be stored in a 'COLUMN'.
- A *reference* MUST identify an *object type instance* represented elsewhere, possibly in another 'table', possibly in the same table, possibly in another document.
- An *attribute* SHOULD be stored in the same table as its containing *object type.*
- etc

Clearly free-form English sentences as the ones in the example are not what we are after. If we want to be able to identify how a data model is represented in some custom serialization we need a formal, computer readable mapping language.

One part of the mapping language should be anchored in a formally defined serialization language. After all, for some tool to interpret a serialization, it MUST understand its format. A completely freeform serialization is not under consideration here. This document assumes the target meta-model is VOTable.

The mapping language must support the interpretation of elements from the serialization language in terms of elements from the data model. If we want to define a generic mapping mechanism, one by which we can describe how a general data model is serialized inside a VOTable, it is necessary to use a general data model *language* as the target for the mapping, such as the one described above. This language can give formal and more explicit meaning to data modeling concepts, possibly independent of specific languages representation languages such as XML schema, Java, or the relational model.

This document uses VO-DML as the target language.

The final ingredient in the mapping language is a mechanism that ties the components from the two different meta-models together into *sentences*. This generally requires some kind of explicit annotation, some meta-data elements that provide an identification of source to target structure. This document uses an extension to VOTable with a VODML element which can provide this link in a rather simple manner.

This solution is sufficient and it is in some sense the simplest and most explicit approach for annotating a VOTable. It may *not* be the most natural or suitable approach for other meta-models such as FITS or TAP_SCHEMA. We discuss this at the end of this document.

4 Mapping with the `VODML` element.

This section summarizes the technical basis of this reccomendation.

The present specification, in conjunction with with VO-DML recommendation (Lemson 2015) provides a formal mechanism for using such data model identifiers, although different from the original `@utype` attribute definition and its usages (Graham 2013).

VOTable 1.2 introduced the `@utype` attribute, which was intended to represent "pointers into a data model". A precise and formal definition on how this *pointing* was to be achieved and a description of its meaning was missing though.

First, a formal definition of the target of the pointers was missing. To solve this, data models were usually accompanied by a list of *utypes* (**TODO** [TBD refer to STC, Characterization, Spectrum?]), and these could be used as values for said `@utype`, be it in VOTable or for example in the Table Access Protocol metadata. These were not linked in any formal, machine readable way to the underlying data model.

Basically it means that the data model is represented solely by a list of attributes, which does not do justice to the complexity of data models describing complex data products like Data Cubes or the provenance of

Simulations. These contain complex object hierarchies organized in graphs with various types of relations between individual objects. It also proved difficult to express the relationship among different, but overlapping, data models, with much discussion centred on the question how to reuse utypes from one model in the definition of another.

The approach is basically not much more than another vocabulary, similar to UCDs ("UCDs" 1900), or SKOS vocabularies ("PR on Use of SKOS Vocabularies in IVOA" 1900), obtained by different means. Efforts were made to provide some structure to these values that might provide some hints of their location in a model, but there was no formal mechanism on how to derive that structure and it was unclear whether it could truly represent the richness of the existing and future data models. In particular there was no standard defined how this could be achieved and no common usage patterns were discovered (Graham 2013).

VO-DML provides a formal target for these pointers in the data model itself and formally defines how models can be reused in the definition of other, dependent models. Precisely *how* to use these pointers in a VOTable to provide a complete annotation useful for interoperability requires more work though.

The current specification provides such a definition. It shows how data publishers can identify also the more complex data model elements such as structured types and relationships inside some published data source, be it a VOTable or relational database published through the TAP protocol.

This specification defines various *mapping patterns* from VOTable to VO-DML. Such a pattern identifies a VOTable element with a VO-DML element. The VO-DML element is said to be *represented* by the VOTable element. The mapping pattern indicates that instances of identified VO-DML types are present in the VOTable. These may be atomic *values* (instances of VO-DML ValueTypes (Lemson 2015)) or represented by cells in a table column identified by a `FIELD`. Alternatively they may be instances of structured types.

The extension to the VOTable schema is reproduced in sec. 7.

4.1 VODMLReference

This XML type represents a reference to a single element in a VO-DML/XML document. It takea over the role of the `@utype` attribute in this regards. Whenever we wish to refer to instances of the VODMLReference type we will call them **vodmlref**-s. A vodmlref is a string with the following syntax:

```
vodmlref ::= prefix ‘’: vodml-id
```

The prefix identifies the model in which the element identified by the suffix is defined.

vodml-ids are always considered opaque, meaning that clients have no reason to parse them. They are identifiers mapping VOTable elements to VO-DML elements in the identified data model. Thus, they must follow the same syntax rules defined in the VO-DML/Schema document.

Prefixes MUST be exactly the same as the **name** attribute of the model in the VO-DML/XML document that defines it. They are sequences of [A-Za-z0-9_-], and they are case sensitive.

For new models, that are not (yet) standardized or for custom data models used in a smaller community, it is recommended to form DM prefixes as `<author-acronym>_<dm-name>`, where the `<dm-name>` is the name of a standard data model; thus, NED's derivation of spec could have `ned_spec` as a prefix, CDS's derivation `cds_spec`.

Prefixes correspond to major versions for the corresponding data models. Thus, **vodmlrefs** remain constant over "compatible" changes in the sense of (Lemson 2015). In consequence, clients must assume a compatible extension when encountering an unknown **vodmlref** with a known prefix (and should in general not fail).

Another consequence of this rule is that there may be several VO-DML URLs for a given prefix. To identify a data model, use the prefix, not the VO-DML URL, which is intended for retrieval of the data model definition exclusively. In case a client requires the exact minor version of the data model, it must inspect the models declarations as described in sec. 6

(**TODO** OL: This doesn't feel right. I believe minor versions should be uniquely identified by a URI and without having to parse the descriptor, especially since we have started talking about registering models in the Registry.)

How to look for a vodmlref in a document (**TODO** to fill out once the syntax has settled)

5 General information about this spec

5.1 Sample model and instances

(**TODO** This needs to be filled with the designated sample model)

5.2 Single-table representations and Object-Relational Mapping

Broadly speaking, this specification is all about Object-Relational Mapping (ORM). Data Models are represented in VO-DML according to an Entity-Relationship paradigm, in a fashion that is implementable by relational databases, object oriented languages, and possibly by certain document oriented data bases as well.

As VOTable can represent several tables in the same file with rich metadata, one can look at VOTable as a data base that can store instances of complex relational models.

Such models are usually defined in terms of entities, with each table representing each entity, and relationships that can be expressed as tables themselves or as constraints on the values in the tables, and most often with a combination of tables and constraints. For instance, a Many-To-Many relationship between two Entities is usually represented in the relational model as a table holding IDs of instances from the tables representing the Entities, with Foreign Keys constraints.

Astronomers mainly work with single tables that hold flattened representations of relatively simple models, although in some cases complex data models are serialized in several tables inside the same file.

This specification covers both requirements. Serializations of simple models in a flattened table are easier to achieve than complex ORM mappings where information is normalized into different tables, but they are both achievable in VOTable. Moreover, the hybrid case of partly de-normalized representations, where the model is only partly normalized, is more challenging but should also be addressable in terms of this specification.

In any case, the examples in sec. 6 are focused on the single-table, flattened representations of instances according to some data model. Some of the patterns described in these sections are also applicable to simple ORM cases. Especially the sections dealing with mapping reference and composition relations also deal with the more complex cases of proper ORM mappings, where data is partly or completely normalized into different tables.

The simple and complex ORM patterns described by this specification usually belong to very different concrete use cases, so it should be acceptable in a broad range of cases that implementers, both on the server and on the client side, focus on the single-table mappings. Data providers requiring more complex patterns, more advanced applications, or applications built on top of standard software libraries that implement this specification as a whole will need to take advantage of the ORM mapping patterns.

6 Patterns for annotating VOTable [NORMATIVE]

In this section we list all legal mapping patterns that can be used to express how instances of VO-DML-defined types are represented in a VOTable and the possible roles they play. It defines the VOTable annotation syntax, what restrictions there are, and how to interpret the annotation semantically.

The organization of the following sections is based on the XML types introduced in the new VOTable schema. The entire VO-DML annotation is included in a new `VODML` element and its descendents. The following sections introduce each element and how it is used to map the VO-DML concepts to VOTable documents.

In particular, secs. 6.1, 6.2 describe the schema elements that provide data model annotations, while sec. 6.3 inverts this approach and shows how to map VO-DML concepts to the VOTable elements that have been introduced.

Some comments on how we refer to VOTable and VO-DML elements:

- When referring to VOTable elements we will use the notation by which these elements will occur in VOTable documents, i.e. in general "all caps", E.g. `GROUP`, `FIELD`, (though `FIELDref`).
- When referring to an XML attribute on a VOTable element we will prefix it with a '@', e.g. `@id`, `@ref`.
- References to VO-DML elements will be CamelCase and in **`bold face`** ↪ , using their VO-DML/XSD type definitions. E.g. **`ObjectType`**, **`Attribute`**.

The following list defines some shorthand phrases (*italicized*), which we use in the descriptions below:

- Generally when using the phrase *meta-type* we mean a "kind of" type as defined in VO-DML. These are **`PrimitiveType`**, **`Enumeration`**, **`DataType`** and **`ObjectType`**.
- With *atomic type* we will mean a **`PrimitiveType`** or an **`Enumeration`** as defined in VO-DML.
- A *structured type* will refer to an **`ObjectType`** or **`DataType`** as defined in VO-DML.
- With a property *available on* or *defined on* a (structured) type we will mean an **`Attribute`** or **`Reference`**, or (in the case of **`ObjectType`**s) a **`Composition`** defined on that type itself, or inherited from one of its base class ancestors.
- A VO-DML **`Type`** *plays a role* in the definition of another (structured) type if the former is the declared data type of a property available on the latter.
- When writing that a VOTable element *represents* a certain VO-DML type, we mean that the VOTable element is mapped either directly to the type, or that it identifies a role played by the type in another type's definition.

- A *descendant* of a VOTable element is an element contained in that element, or in a descendant of that element. This is a standard recursive definition and can go up the hierarchy as well: an *ancestor* of an element is the direct container of that element, or an ancestor of that container.

When we say this section is NORMATIVE we mean that:

1. when a client finds an annotation pattern conforming to one defined here, that client is justified in interpreting it as described in the comments for that pattern. It is an ANNOTATION ERROR if that were to lead to inconsistencies[4].
2. when a client encounters a pattern not in this list, the client SHOULD ignore it. Interpreting it as a mapping to a data model MAY work, but is not mandated and other clients need not conform to this.

6.1 The `VODML` element

This specification introduces a new `VODML` element to `VOTABLE`.

The element is a direct child of `VOTABLE` and only one `VODML` element is allowed in each file. This element contains all the data model annotations for the entire file.[5]

There are three sections making up the contents of the `VODML` annotation:

- model declarations (`MODEL` elements)
- global instances (`GLOBALS` elements)
- tabular instance templates (`TEMPLATES`).

[4]E.g. when interpreting an `<INSTANCE>` as a certain **`ObjectType`**, if one of its children is not annotated or identifies a child element that is not available on the type, this is an error. For each pattern there is a set of rules that, if broken, are annotation errors. (**TODO** we better strive to make these comprehensive. OL thinks this is too strict of a default rule. We should probably be more lenient by default and add strictness where required. A lot is already mandated by the new schema.)

[5]Earlier versions of this specification had the `VODML` element defined as a potential child of several existing elements, including `RESOURCE` and `TABLE`. While this achieved a form of locality by bringing the annotations close to the elements it was annotating, it introduced two different, valid ways of annotating files. Also, the locality it achieved was completely lost in the more complex cases with multiple `TABLE` elements, where this kind of locality matters the most. A single instance of `VODML` is much simpler to implement for data providers and consumers alike, and it provides only one obvious way of annotating a file. On the other hand, the single `VODML` element might pose challenges to providers streaming VOTable instances without a knowledge of the semantics of the contents upfront. This case seems more like a theoretical possibility than an actual implementation at this point, so it was ignored. It could always be possible to add the `VODML` element as a child of other elements in future versions of this specification, keeping into account actual requirements from actual implementations rather than just a theoretical possibility.

The model declarations introduce the models used in the annotation. Global instances are direct instances, i.e. instances whose values are completely determined by the annotation, as opposed to instance templates, which annotates multiple instances that have some of their values represented in tabular format inside the annotated file, as described in more detiail in the following sections.

6.1.1 Example

Listing 1: Example VOTable with basic annotations. This VOTable illustrates the basic annotation elements in the 'VODML' section. More complex examples will be provided in the following sections.

```xml
<?xml version="1.0" encoding="UTF-8"?><VOTABLE xmlns="http://www.ivoa.net/xml/VOTable/v1.4" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <VODML>
    <MODEL>
      <NAME>ivoa</NAME>
      <URL>http://volute.g-vo.org/svn/trunk/projects/dm/vo-dml/models/ivoa/IVOA.vo-dml.xml</URL>
    </MODEL>
    <MODEL>
      <NAME>filter</NAME>
      <URL>http://volute.g-vo.org/svn/trunk/projects/dm/vo-dml/models/sample/filter/Filter.vo-dml.xml</URL>
    </MODEL>
    <MODEL>
      <NAME>sample</NAME>
      <URL>https://raw.githubusercontent.com/olaurino/jovial/new-mapping/src/test/resources/votable/Sample.vo-dml.xml</URL>
    </MODEL>
    <GLOBALS>
      <INSTANCE dmtype="sample:catalog.SkyCoordinateFrame" ID="_icrs" >
        <ATTRIBUTE dmrole="sample:catalog.SkyCoordinateFrame.name">
          <LITERAL value="ICRS" dmtype="ivoa:string"/>
        </ATTRIBUTE>
      </INSTANCE>
    </GLOBALS>
    <TEMPLATES tableref="_table1">
      <INSTANCE dmtype="sample:catalog.SDSSSource">
        <ATTRIBUTE dmrole="sample:catalog.AbstractSource.position">
          <INSTANCE dmtype="sample:catalog.SkyCoordinate">
            <REFERENCE dmrole="sample:catalog.SkyCoordinate.frame">
              <IDREF>_icrs</IDREF>
            </REFERENCE>
            <ATTRIBUTE dmrole="sample:catalog.SkyCoordinate.longitude"
                ↪ >
              <COLUMN dmtype="ivoa:RealQuantity" ref="_ra"/>
            </ATTRIBUTE>
            <ATTRIBUTE dmrole="sample:catalog.SkyCoordinate.latitude">
              <COLUMN dmtype="ivoa:RealQuantity" ref="_dec"/>
            </ATTRIBUTE>
          </INSTANCE>
        </ATTRIBUTE>
        <ATTRIBUTE dmrole="sample:catalog.AbstractSource.name">
          <COLUMN dmtype="ivoa:string" ref="_designation"/>
        </ATTRIBUTE>
      </INSTANCE>
    </TEMPLATES>
  </VODML>
  <RESOURCE>
    <TABLE ID="_table1">
      <FIELD datatype="float" arraysize="1" ID="_ra"/>
      <FIELD datatype="float" arraysize="1" ID="_dec"/>
      <FIELD datatype="char" arraysize="16" ID="_designation"/>
      <DATA>
        <TABLEDATA>
          <TR>
            <TD>122.992773</TD>
            <TD>-2.092676</TD>
            <TD>08115826-0205336</TD>
          </TR>
          <TR>
            <TD>122.986794</TD>
            <TD>-2.095231</TD>
            <TD>08115683-0205428</TD>
          </TR>
          <TR>
            <TD>123.033734</TD>
            <TD>-2.103671</TD>
            <TD>08120809-0206132</TD>
          </TR>
        </TABLEDATA>
      </DATA>
    </TABLE>
  </RESOURCE>
</VOTABLE>
```

The example in lst. 1 uses sample models that are simple enough to be useful for exploring the mapping patterns, but also complete as they exercise the entirety of the mapping schema. These models are not IVOA standard models themselves, but were defined for illustrating the mapping specification only.

6.1.2 Models Declaration: `MODEL`

A VOTable can provide serializations for an arbitrary number of data model types. In order to declare which models are represented in the file, data providers **must** declare them through the `MODEL` elements.

Only models that are used in the file must be declared. A model is used if at least one element in the file has a `vodml-ref` with the model's prefix/name. In other terms, only models that define `vodml-ids` used in the annotation (see sec. 4.1) must be declared.

A `MODEL` is uniquely and globally identified by its `NAME`, which refers to the latest available minor version of a specific model. According to the VO-DML specification (Lemson 2015), all minor versions of a model are compatible with each other, and major versions have different names (e.g. `stc` vs `stc2`).

Clients should not be sensible to the specific minor version of a model and can simply ignore `vodml-refs` they are not familiar with.

The `NAME` is also the prefix of the `vodml-refs` pointing to elements of a specific model.

For clients that need to parse the model descriptions, i.e. the VODM-L/XML file of a model, data providers **must** include the URL of the VODM-L/XML document in the `MODEL/URL` element. if the model is defined in an IVOA recommendation, then this URL **must** be persistent, and the same URL that is registered for that model in the IVOA registries. If the model is a custom extensions, it **should** also be registered in IVOA registries. However, since this is not a requirement, for custom extensions that are not registered data providers **must** still provide the URL of the VODML/XML description file, and care should be taken into ensuring that the URL is persistent, otherwise clients will not be able to read the model definition. For more information on registering and identifying data models, please refer to the VO-DML specification ((Lemson 2015))

Note that because of the above procurements, clients are not required to resolve the `NAME` of the model to its VODML/XML document, but can, if necessary, rely on the `URL` element.

In lst. 1 three models are declared, with the names `ivoa`, `filter`, and `sample`, which are the prefixes used in all the `vodml-refs` in the file.

6.1.3 Instance Annotation: `INSTANCE`

VO-DML structured types are annotated by using the new `INSTANCE` XML element. Note that there is no difference, from a schema point of view, between `**ObjectType**`s and `**DataType**`s. However, some restrictions apply and are enforced through schematron rules. More details are provided in secs. 6.3.3, 6.3.4.

Instances **must** a *type* (`@dmtype` attribute). Note that instances are not annotated with a role, as roles are mapped to specific XML elements (see sec. 6.2). This is compatible with the intuitive fact that the same instance can have different roles in relationships with other instances.

Instances **may** be provided with an `@ID` value which allows other objects to refer to the instance itself (see sec. 6.2.6).

Instances usually have a number of *roles*, which can be themselves instances through a composition relationship (sec. 6.2.9), references to other instances (sec. 6.2.5), or attributes (sec. 6.2.1).

As explained in more detail in sec. 6.2.9 and related sections, a composition relationship is annotated so that parents contain or refer to children instances and children point to their parent instances through the `CONTAINER` element (sec. 6.2.11).

6.1.4 Global Instances: `GLOBALS`

Some annotations may map the VOTable contents to instances of data model types that are global in the file, possibly because such instances are referenced by other intances that annotate specific tables. More generally, some annotations will define instances that are completely defined in terms of constant value, i.e. they are not represented in tabular form. Rather, they are completely and directly represented by an `XML` element.

Such instances should be included in the `GLOBALS` element.

An annotation can have multiple `GLOBALS` sections so to group together global instances that are related to each other.

In particular, `GLOBALS` sections can be identifier by an `@ID` attribute. This is particularly useful when referencing global instances from table elements, see for instance sec. 6.2.8.

`GLOBALS` **must** only contain direct representations of instances, i.e. `INSTANCE` elements that do not have any `COLUMN` elements directly or in any of their descendants. This rule is not enforces via the XSD schema but by schematron rules.

Also, `GLOBALS` **should not** contain any `INSTANCE`s with `REFERENCE`s to indirect `INSTANCE`s, unless the model allows for multiple references with the same role, which is however deprecated by the `VO-DML` specification.

As an example, lst. 1 contains one `GLOBALS` section with one instance representing a coordinate frame in the rather simplistic *sample* model. The coordinate frame instance is completely determined by the global `INSTANCE`.

6.1.5 Tabular Instances: `TEMPLATES`

Most data in VOTable is expressed in tabular form, each row representing one individual instance. When this is the case, instances need to be represented through *templates* in the `TEMPLATES` section of the annotation. An `INSTANCE` in a template is identical to `INSTANCE`s in the `GLOBALS` section, but they are also allowed to contain COLUMN elements (see sec. 6.2.4), which are references to regular VOTable FIELDs. Rather than being directly and completely represented by its INSTANCE annotation, instances are thus materialized for each row in a table, according to their annotation. Note that a COLUMN may be present either directly in an INSTANCE or in any of its descendent INSTANCEs.

TEMPLATES **must** refer to a TABLE through the tableref attribute. COLUMN elements inside a template INSTANCE **must** be references to FIELDs in the TABLE identified by tableref.

In lst. 1 table _table1 contains simple astronomical sources with columns for the sources names, Right Ascension, and Declination in the ICRS reference frame. Thus, the sources are annotated using an INSTANCE element inside the TEMPLATES section of the VODML element. The instance and its attributes refer to the three columns in the table and map their contents to the attributes of sources and positions defined in the respective data model.

6.1.6 Direct vs Indirect Instances

As an illustration of the difference between direct and indirect instances, consider the following VOTable:

Listing 2: VOTable where all instances are direct rather than indirect.

```
<?xml version="1.0" encoding="UTF-8"?><VOTABLE xmlns="http://www.
    ↪ ivoa.net/xml/VOTable/v1.4" xmlns:xsi="http://www.w3.org/2001/
    ↪ XMLSchema-instance">
  <VODML>
    <MODEL>
      <NAME>ivoa</NAME>
      <URL>http://volute.g-vo.org/svn/trunk/projects/dm/vo-dml/models
          ↪ /ivoa/IVOA.vo-dml.xml</URL>
    </MODEL>
    <MODEL>
      <NAME>filter</NAME>
      <URL>http://volute.g-vo.org/svn/trunk/projects/dm/vo-dml/models
          ↪ /sample/filter/Filter.vo-dml.xml</URL>
    </MODEL>
    <MODEL>
      <NAME>sample</NAME>
      <URL>https://raw.githubusercontent.com/olaurino/jovial/new-
          ↪ mapping/src/test/resources/votable/Sample.vo-dml.xml</
          ↪ URL>
    </MODEL>
    <GLOBALS>
      <INSTANCE dmtype="sample:catalog.SkyCoordinateFrame" ID="_icrs"
          ↪ >
        <ATTRIBUTE dmrole="sample:catalog.SkyCoordinateFrame.name">
          <LITERAL value="ICRS" dmtype="ivoa:string"/>
        </ATTRIBUTE>
      </INSTANCE>
      <INSTANCE dmtype="sample:catalog.SDSSSource">
        <ATTRIBUTE dmrole="sample:catalog.AbstractSource.name">
          <LITERAL value="08120809-0206132" dmtype="ivoa:string"/>
        </ATTRIBUTE>
        <ATTRIBUTE dmrole="sample:catalog.AbstractSource.position">
          <INSTANCE dmtype="sample:catalog.SkyCoordinate">
            <ATTRIBUTE dmrole="sample:catalog.SkyCoordinate.longitude"
                ↪ >
              <LITERAL value="123.033734" dmtype="ivoa:RealQuantity"/>
            </ATTRIBUTE>
            <ATTRIBUTE dmrole="sample:catalog.SkyCoordinate.latitude">
              <LITERAL value="-2.103671" dmtype="ivoa:RealQuantity"/>
            </ATTRIBUTE>
            <REFERENCE dmrole="sample:catalog.SkyCoordinate.frame">
              <IDREF>_icrs</IDREF>
            </REFERENCE>
          </INSTANCE>
        </ATTRIBUTE>
      </INSTANCE>
      <INSTANCE dmtype="sample:catalog.SDSSSource">
        <ATTRIBUTE dmrole="sample:catalog.AbstractSource.name">
          <LITERAL value="08115683-0205428" dmtype="ivoa:string"/>
        </ATTRIBUTE>
        <ATTRIBUTE dmrole="sample:catalog.AbstractSource.position">
          <INSTANCE dmtype="sample:catalog.SkyCoordinate">
            <ATTRIBUTE dmrole="sample:catalog.SkyCoordinate.longitude"
                ↪ >
              <LITERAL value="122.986794" dmtype="ivoa:RealQuantity"/>
            </ATTRIBUTE>
            <ATTRIBUTE dmrole="sample:catalog.SkyCoordinate.latitude">
              <LITERAL value="-2.095231" dmtype="ivoa:RealQuantity"/>
            </ATTRIBUTE>
            <REFERENCE dmrole="sample:catalog.SkyCoordinate.frame">
              <IDREF>_icrs</IDREF>
            </REFERENCE>
          </INSTANCE>
        </ATTRIBUTE>
      </INSTANCE>
      <INSTANCE dmtype="sample:catalog.SDSSSource">
        <ATTRIBUTE dmrole="sample:catalog.AbstractSource.name">
          <LITERAL value="08120809-0206132" dmtype="ivoa:string"/>
        </ATTRIBUTE>
        <ATTRIBUTE dmrole="sample:catalog.AbstractSource.position">
          <INSTANCE dmtype="sample:catalog.SkyCoordinate">
            <ATTRIBUTE dmrole="sample:catalog.SkyCoordinate.longitude"
                ↪ >
              <LITERAL value="122.992773" dmtype="ivoa:RealQuantity"/>
            </ATTRIBUTE>
            <ATTRIBUTE dmrole="sample:catalog.SkyCoordinate.latitude">
              <LITERAL value="-2.092676" dmtype="ivoa:RealQuantity"/>
            </ATTRIBUTE>
            <REFERENCE dmrole="sample:catalog.SkyCoordinate.frame">
              <IDREF>_icrs</IDREF>
            </REFERENCE>
          </INSTANCE>
        </ATTRIBUTE>
      </INSTANCE>
    </GLOBALS>
   </VODML>
  </VOTABLE>
```

This is a representation of the very same instances as in lst. 1. However, while lst. 1 uses template instances in `TEMPLATES` and a `TABLE` element to represent the values of the sources, lst. 2 uses direct representations for all three sources in the example.

Generally speaking, this specification is used to annotate data products according to one or more data models. In this sense, the choice on whether to use direct or indirect representations is driven by the data being produced. In principle, for each indirect representation there is an equivalent indirect representation where template instances and one or more tables are turned into a number of explicit, direct instances.

6.1.7 Schema Constraints

In order to be valid, the `VODML` **must** contain at least one `MODEL` instance and at least one of either `GLOBALS` or `TEMPLATES`, which in turn **must** contain at least one `INSTANCE`.

6.2 Relations

All instances have a type, and some instances also play a *role* in another instance. For example, if a **Source** has a **position** of type **Coordinate**, then an instance of type **Coordinate** will also have the role of **position** in the enclosing **Source** instance.

The role corresponds to a relationship between two types.

The following roles are mapped from VO-DML to VOTable:

- Attributes (sec. 6.2.1).
- Compositions (sec. 6.2.9).
- References (sec. 6.2.5).

6.2.1 Attributes: `ATTRIBUTE`

An attribute may be represented by another instance (sec. 6.1.3), by a primitive or enumerated value (secs. 6.2.2, 6.2.3), or by a column (sec. 6.2.4) if the parent instance is a template, i.e. an indirect representation.

An `ATTRIBUTE` **must** have a `@dmrole` attribute indicating the role of the attribute, and such `@dmrole` **must** be a valid vodml-ref as defined in one of the VO-DML models declared in the models section (sec. 6.1.2). The vodml-ref **must** also identify a VO-DML Attribute.

6.2.2 Attributes: `LITERAL`

A `LITERAL` **must** represent an attribute with a primitive or enumerated type. It **must** have a `@value` attribute and optionally a `@unit`. It **must** also have a `@dmrole` attribute, whose value **must** be a valid vodml-ref defined in one of the declared models (sec. 6.1.2) and identify a VO-DML role with a primitive type.

6.2.3 Attributes: `CONSTANT`

A `CONSTANT` is a reference to a VOTable `PARAM`. This element can be used in place of a `LITERAL` to point to an existing `PARAM` in the same VOTable, avoiding duplicated values.

Note that the target of the reference **must** be a `PARAM`.[6]

6.2.4 Template Attributes: `COLUMN`

A `COLUMN` is a reference to a VOTable `FIELD`. Instances representing attributes as `COLUMN`s **must** be defined inside a `TEMPLATES` element. A `COLUMN` ↪ **must** have a `@dmtype` attribute whose value **must** be a valid vodml-ref defined by one of the declared models (sec. 6.1.2). Such vodml-ref **must** identify the VO-DML type corresponding to the enclosing attribute's role.

[6]There are a few design considerations behind the choice of not including `PARAM`s, `PARAMref`s and `FIELDref`s directly in this specification. One is that the VO-DML mapping schema might be used outside of is VOTable context to describe and map data model instances in XML. The other is to keep the VOTable and VO-DML schemata as much decoupled as possible, in order to facilitate future revisions. Also, `PARAM` is too rich of an element with its own attributes and semantics. Such semantics are rather different than the ones in `VO-DML`, in particular regarding the different kinds of types (`@utype`, `@xtype`, `@datatype`) and annotations (`@arraysize`, `@ucd`) that a `PARAM` is composed of, which do not have an equivalent in VO-DML.

6.2.5 References: `REFERENCE`

A reference is a relationship between a referring instance and a referred instance. While the referring instance can be both an `ObjectType` and a `DataType` the referred instance **must** be of an `ObjectType`.

The reference can be to an identified direct instance in the same file (sec. 6.2.6), to a remote instance identified by a URI (sec. 6.2.7), or to an instance indirectly represented by a template, through a foreign key pointing at the referred instance's primary key (sec. 6.2.8).

6.2.6 References: `IDREF`

In order to use `IDREF` the referred instance **must** be an instance of an `ObjectType`, it **must** be serialized as a direct instance, and it **must** be located in the same document as the referrer instance. Also, there **must** be a correspondent relationship defined in the VO-DML description of the model in which the referrer type is defined.

6.2.7 References: `REMOTEREFERENCE`

In order to use `REMOTEREFERENCE` the referred instance **must** be an instance of an `ObjectType`, it **must** be serialized as a direct instance, and it **must** be located in a different document than the referrer instance. Also, there **must** be a correspondent relationship defined in the VO-DML description of the model in which the referrer type is defined.

`REMOTEREFERENCE` is of type `xs:anyURI` an **must** identify a globally unique instance through a specific URI (e.g. an IVOA Resource Name), **or** a URL to the serialization of the instance, but the URL **must** be persistent.

6.2.8 References: `FOREIGNKEY`

The referred instance may be serialized as a row in a table different than the one serializing the referrer (if any). In this case the reference is annotated through a `FOREIGNKEY` (sec. 6.2.13).

6.2.9 Composition: `COMPOSITION`

A composition is a whole-part relationship between `ObjectTypes`, where one instance is said to be the container, or parent, or whole, and the other is said to be the contained, or child, or part. In a VOTable composed instances may be serialized directly within the parent instance (sec. 6.1.3) if the parent is directly representated or if the part is serialized in the same table as the parent, or externally (sec. 6.2.10) if the part is indirectly represented and serialized in a different table. Children refer back to their parent through an implicit container relationship (sec. 6.2.11). The reference is achieved through a foreign key (sec. 6.2.13) referencing the container's primary key (sec. 6.2.12).

In order to be valid, the composition **must** link two `ObjectType`s and there must be a correspondent relationship defined in the VO-DML description of the model. The number of contained instances must be compatible with the relationship's multiplicity.

The `COMPOSITION` element **must** have a `@dmrole` attribute, whose value **must** be a valid vodml-ref identifying a composition relationship in one of the models declared (sec. 6.1.2), and types of the parent and children **must** be compatible with the relationship defined in that model.

6.2.10 Composition: `EXTINTANCES`

A composition relationship can delegate the definition of some of the children to an external `INSTANCE` declaration elsewhere in the file, for example if the instances are defined in a different table.

The `EXTINSTANCES` element is useful for clients in that it links instances from the parent to the children. In relational data bases, the composition relatioship is usually implementing by referencing objects from the parts to the whole, not the other way around (see @[sec:norm-container]). This specification provides a way for containers to declare where the contained objects are located and described.

(**TODO** this leaves the door open to the possibility that children refer to their parents but not the other way around. Should clients be prepared to this possibility? If so, then the `EXTINSTANCES` element is redundant. Otherwise what seems to be redundant is the `CONTAINER` element).

6.2.11 Composition: `CONTAINER`

On the part side of a composition relationship, especially in complex Object Relational Mapping applications, parts include a reference to their containers. This is consistent with the relational implementation of the composition relationship. For more details see sec. 6.2.13.

6.2.12 `PRIMARYKEY`

Instances **may** be identified by an object identifier through the `PRIMARYKEY` element.

An object identifier can have any number of `PKFIELD` fields. Each `PKFIELD` is a choice among a `LITERAL`, i.e. a local value (sec. 6.2.2), a `COLUMN`, i.e. a reference to a cell value (sec. 6.2.4), or a `CONSTANT`, i.e. a reference to a `PARAM` (sec. 6.2.3).

Fields are values that make up the object identifier. Generally only one value is used, but it is not uncommon for objects to be identified by a tuple of values.

Primary keys **must** be unique within object types. In other terms there can be no two instances with the same primary key in a single file. If two instances appear to have the same primary key then client's behavior is undefined.

6.2.13 `FOREIGNKEY`

In Object Relational Mapping a foreign key is the mechanism by which instances in a table identify other instances in many-to-one and many-to-many relationships with itself.

In a simple model where a source can be observed in an arbitrary number of photometric filters, one would usually have a table for sources, with IDs and general metadata, a table for photometric filters, and a table for luminosity measurements of a source. In this implementation, the Luminosity entity is in a many-to-one relationship with both the Source and the Filter tables. This relationship is implemented by providing the Luminosity table with two foreign key columns to the Source and Luminosity tables. The value of the foreign keys is the ID of the Source and Filter instances to which each luminosity measurements refer to.

A query might query the data base for "all the luminosity measurements of Source 3c273", implying the ownership relationship of a Source with its Luminosities, although there is no actual reference from the Source table to the Luminosity table.

Providing the parent table with a column for each reference to an arbitrary number of potential children is clearly unsustainable, if at all possible.

A `FOREIGNKEY` can provide a `TARGETID` to point to an identified element in the same file. (**TODO** Be more specific here on how to use `TARGETID`)

A `FOREIGNKEY` **must** contain at least one `PKFIELD`, and the structure of the key fields must be compatible with the one of the primary keys of the referenced instance.

6.3 Representing Types

6.3.1 `PrimitiveType`

Primitive Types are types without structure, and so instances are represented by a simple value. They can be mapped to:

- a `LITERAL` element if the value is provided in the file's header (sec. 6.2.2);

- a `CONSTANT`, i.e. a reference to a `PARAM`, if the value is mapped to the value of an existing `PARAM` (sec. 6.2.3);
- a `COLUMN`, i.e. a reference to a `FIELD`, if the value is in a table cell. In this case we say that the enclosing instance is indirectly represented by an `INSTANCE` template, with actual instances serialized in tabular format (sec. 6.2.4).

6.3.2 `Enumeration` and `EnumerationLiteral`

Enumerations are primitive types with a limited number of possible, enumerated values. Enumerations are mapped to the same elements as primitive types (sec. 6.3.1), with the limitation that values **must** be valid enumeration literals compatible with the type declared by the enclosing `ATTRIBUTE` element.

6.3.3 `ObjectType`

Object types are mapped to the `INSTANCE` element (sec. 6.1.3). Object types can have **`DataType`**d attributes, which are mapped to the `ATTRIBUTE` element (sec. 6.2.1). The `ATTRIBUTE` **must** have a **`@dmrole`** corresponding to the model-defined role identifier. In turn, the `ATTRIBUTE` **must** contain an `INSTANCE` with a type compatible with the attribute's role. Note that the type can be the type declared directly in the model or any specialization of that type defined in any model declared via the `MODEL` element.

Object types can also have attributes with a **`PrimitiveType`** type or with an **`Enumeration`** type. In this case attributes are mapped following the patterns described in sec. 6.3.1 and sec. 6.3.2 respectively.

Object types can hold references to instances of other object types. This roles are mapped to the `REFERENCE` element (sec. 6.2.5). In this case the `REFERENCE` **must** have a **`@dmrole`** corresponding to the model-defined reference relationship.

Finally, object types can be in composition relationships with other object types. An object type can be both the *parent*, or *whole*, and *contain* its *children*, or *parts* (sec. 6.2.9).

A parent `INSTANCE` will have a `COMPOSITION` element with the **`@dmrole`** of the composition relationship defined in the model.

In addition to `INSTANCE`s, parents can also point to instances described elsewhere in the document through the `EXTINSTANCE` elements.

For each composition relationship an arbitrary number of instances may be present, with two limitations:

- the number of instances must be compatible with the multiplicity of the relationship;

- the instances must have `@dmtypes` compatible with the data type of the relationship defined in the model, i.e. the exact type declared there or any of its subtypes.

A children `INSTANCE` will have a `CONTAINER` element pointing at the

As described in sec. 6.1.4 and sec. 6.1.5 an **`ObjectType`** can be represented both directly or indirectly through standalone instances or instance templates respectively.

6.3.4 `DataType`

Data types are mostly mapped like object types. This removes a lot of burden from data providers and clients that do not need to validate datasets, simplifying the syntax.

Some constaints apply to **`DataType`**s. They are not enforced through XSD but datasets **must** meet such constraints in order to validate.

VO-DML **`DataTypes`** **cannot** be in composition relationship with any other types, so `COMPOSITION` and `CONTAINER` **cannot** be used inside an `INSTANCE` representing a **`DataType`**. Similarly, **`DataType`**s cannot have a `PRIMARYKEY`.

Also, there **cannot** be references pointing to a **`DataType`** instance.

6.3.5 Type Generalization and Inheritance

(**TODO This needs to be expanded and completed with examples)

VO-DML allows types to *extend* or *specialize* other types. As it happens in object oriented languages, an instance of a specialization, or *subtype* of a more general type, or *supertype*, is also a valid instance of the supertype, and can be used wherever an instance of a supertype is expected.

This means that clients of the supertype must be able to recognize instances of the supertype even though a specialized subtype might have been instantiated. This case includes all kinds of clients discussed in sec. 2.4. What we called *guru* clients can read the VO-DML description files and can figure out the generalization relationships. *Advanced* clients might or might not care about generalizations, but generalization must be mapped with particular care so that *simple* clients can find the information they are seeking, even when a type they are interested in is present as an instance of one of its subtypes.

Since we currently provide only one `@dmtype` per `INSTANCE`, only one type can be expressed for each instance. This type **must** always be the actual type of the instance. This means that clients **cannot** rely on the instance's type in order to recognize instances, unless they rely on VO-DML-aware specialized software libraries.

In VO-DML specialized types inherit all of the supertype's vodml-id descriptors, including the prefix. This means that `@dmroles` can always be matched, whether or not an instance represents a supertype or one of its subtypes.

6.3.6 Quantities

(**TODO** This is a placeholder for specific mapping of ivoa quantities, but we need the ivoa model to settle)

6.3.7 Comparison of `EXTINSTANCES` and `INSTANCE`

The distinction between `INSTANCE` and `EXTINSTANCE` as children of the `COMPOSITION` element allows for common object relational mappings to be used. In a completely normalized implementation each type may be serialized in its own table. However, in practice astronomical archives and datasets show some form of flattening, i.e. some types are serialized as part of the table that represent the parent type, or multiple instances of the part type are serialized in the same table. This is the case, for instance, of sources and their luminosities.

In a mission's archive, and in the data sets they serve, usually a single table represents a number of photometric measurements in different filters. This corresponds to a VO-DML annotation with an indirect `INSTANCE` for the Source type with a `COMPOSITION` element containing a number of indirect `INSTANCE`s for the luminosity measurements. Such instances correpond to the finite set of filters (and errors, and other kinds of metadata) specific to a particular mission. All the `INSTANCE`s, in this example, would point to columns in the same table.

However, consider now the case of a data set that contains a number of sources, each with an arbitrary number of photometric measurements coming from different missions and archives. In this case a single flattened table is not an efficient or effective way of serializing such instances, and multiple tables can be used for sources and luminosities. The `EXTINSTANCES` mechanism provides a mechanism for representing this pattern. (**TODO** However, the safest and most efficient way is to just have children refer to their containers.)

6.4 Full Example

Listing 3: Example covering all of the XML elements and mapping patterns, including multi-table Object Relational Mapping.

```
<?xml version="1.0" encoding="UTF-8"?><VOTABLE xmlns="http://www.
    ↪ ivoa.net/xml/VOTable/v1.4" xmlns:xsi="http://www.w3.org/2001/
    ↪ XMLSchema-instance">
```

```
<VODML>
  <MODEL>
    <NAME>ivoa</NAME>
    <URL>http://volute.g-vo.org/svn/trunk/projects/dm/vo-dml/models
        ↪ /ivoa/IVOA.vo-dml.xml</URL>
  </MODEL>
  <MODEL>
    <NAME>filter</NAME>
    <URL>http://volute.g-vo.org/svn/trunk/projects/dm/vo-dml/models
        ↪ /sample/filter/Filter.vo-dml.xml</URL>
  </MODEL>
  <MODEL>
    <NAME>sample</NAME>
    <URL>https://raw.githubusercontent.com/olaurino/jovial/new-
        ↪ mapping/src/test/resources/votable/Sample.vo-dml.xml</
        ↪ URL>
  </MODEL>
  <GLOBALS>
    <INSTANCE dmtype="sample:catalog.SkyCoordinateFrame" ID="_icrs"
        ↪ >
      <ATTRIBUTE dmrole="sample:catalog.SkyCoordinateFrame.name">
        <LITERAL value="ICRS" dmtype="ivoa:string"/>
      </ATTRIBUTE>
    </INSTANCE>
    <INSTANCE dmtype="filter:PhotometryFilter" ID="_2massH">
      <PRIMARYKEY>
        <PKFIELD>
          <LITERAL dmtype="ivoa:string" value="_2massH"/>
        </PKFIELD>
      </PRIMARYKEY>
      <ATTRIBUTE dmrole="filter:PhotometryFilter.name">
        <LITERAL value="2mass:H" dmtype="ivoa:string"/>
      </ATTRIBUTE>
    </INSTANCE>
    <INSTANCE dmtype="filter:PhotometryFilter" ID="_2massJ">
      <PRIMARYKEY>
        <PKFIELD>
          <LITERAL dmtype="ivoa:string" value="_2massJ"/>
        </PKFIELD>
      </PRIMARYKEY>
      <ATTRIBUTE dmrole="filter:PhotometryFilter.name">
        <LITERAL value="2mass:J" dmtype="ivoa:string"/>
      </ATTRIBUTE>
    </INSTANCE>
    <INSTANCE dmtype="filter:PhotometryFilter" ID="_2massK">
      <PRIMARYKEY>
        <PKFIELD>
          <LITERAL dmtype="ivoa:string" value="_2massK"/>
        </PKFIELD>
      </PRIMARYKEY>
      <ATTRIBUTE dmrole="filter:PhotometryFilter.name">
        <LITERAL value="2mass:K" dmtype="ivoa:string"/>
      </ATTRIBUTE>
    </INSTANCE>
  </GLOBALS>
  <GLOBALS ID="_SDSS_FILTERS">
    <INSTANCE dmtype="filter:PhotometryFilter">
      <PRIMARYKEY>
        <PKFIELD>
          <LITERAL dmtype="ivoa:string" value="sdss:g"/>
        </PKFIELD>
      </PRIMARYKEY>
      <ATTRIBUTE dmrole="filter:PhotometryFilter.name">
        <LITERAL value="sdss:g" dmtype="ivoa:string"/>
      </ATTRIBUTE>
    </INSTANCE>
    <INSTANCE dmtype="filter:PhotometryFilter">
      <PRIMARYKEY>
        <PKFIELD>
          <LITERAL dmtype="ivoa:string" value="sdss:r"/>
        </PKFIELD>
      </PRIMARYKEY>
      <ATTRIBUTE dmrole="filter:PhotometryFilter.name">
        <LITERAL value="sdss:r" dmtype="ivoa:string"/>
      </ATTRIBUTE>
    </INSTANCE>
    <INSTANCE dmtype="filter:PhotometryFilter">
      <PRIMARYKEY>
        <PKFIELD>
          <LITERAL dmtype="ivoa:string" value="sdss:u"/>
        </PKFIELD>
      </PRIMARYKEY>
      <ATTRIBUTE dmrole="filter:PhotometryFilter.name">
        <LITERAL value="sdss:u" dmtype="ivoa:string"/>
      </ATTRIBUTE>
    </INSTANCE>
  </GLOBALS>
  <TEMPLATES tableref="_table1">
    <INSTANCE dmtype="sample:catalog.SDSSSource" ID="_source">
      <PRIMARYKEY>
        <PKFIELD>
          <COLUMN dmtype="ivoa:string" ref="_designation"/>
        </PKFIELD>
      </PRIMARYKEY>
      <ATTRIBUTE dmrole="sample:catalog.AbstractSource.name">
        <COLUMN dmtype="ivoa:string" ref="_designation"/>
      </ATTRIBUTE>
      <ATTRIBUTE dmrole="sample:catalog.AbstractSource.position">
```

```
     <INSTANCE dmtype="sample:catalog.SkyCoordinate">
      <REFERENCE dmrole="sample:catalog.SkyCoordinate.frame">
       <IDREF>_icrs</IDREF>
      </REFERENCE>
      <ATTRIBUTE dmrole="sample:catalog.SkyCoordinate.longitude"
        ↪ >
       <COLUMN dmtype="ivoa:RealQuantity" ref="_ra"/>
      </ATTRIBUTE>
      <ATTRIBUTE dmrole="sample:catalog.SkyCoordinate.latitude">
       <COLUMN dmtype="ivoa:RealQuantity" ref="_dec"/>
      </ATTRIBUTE>
     </INSTANCE>
    </ATTRIBUTE>
    <ATTRIBUTE dmrole="sample:catalog.AbstractSource.
      ↪ positionError">
     <INSTANCE dmtype="sample:catalog.AlignedEllipse">
      <ATTRIBUTE dmrole="sample:catalog.AlignedEllipse.longError
        ↪ ">
       <LITERAL value="0.1" dmtype="ivoa:real"/>
      </ATTRIBUTE>
      <ATTRIBUTE dmrole="sample:catalog.AlignedEllipse.latError"
        ↪ >
       <LITERAL value="0.1" dmtype="ivoa:real"/>
      </ATTRIBUTE>
     </INSTANCE>
    </ATTRIBUTE>
    <COMPOSITION dmrole="sample:catalog.AbstractSource.luminosity
      ↪ ">
     <INSTANCE dmtype="sample:catalog.LuminosityMeasurement">
      <ATTRIBUTE dmrole="sample:catalog.LuminosityMeasurement.
        ↪ type">
       <LITERAL value="magnitude" dmtype="sample:catalog.
         ↪ LuminosityType"/>
      </ATTRIBUTE>
      <REFERENCE dmrole="sample:catalog.LuminosityMeasurement.
        ↪ filter">
       <IDREF>_2massH</IDREF>
      </REFERENCE>
      <ATTRIBUTE dmrole="sample:catalog.LuminosityMeasurement.
        ↪ value">
       <COLUMN dmtype="ivoa:RealQuantity" ref="_magH"/>
      </ATTRIBUTE>
      <ATTRIBUTE dmrole="sample:catalog.LuminosityMeasurement.
        ↪ error">
       <COLUMN dmtype="ivoa:RealQuantity" ref="_errH"/>
      </ATTRIBUTE>
     </INSTANCE>
     <INSTANCE dmtype="sample:catalog.LuminosityMeasurement">
```

```
            <ATTRIBUTE dmrole="sample:catalog.LuminosityMeasurement.type">
              <LITERAL value="magnitude" dmtype="sample:catalog.LuminosityType"/>
            </ATTRIBUTE>
            <REFERENCE dmrole="sample:catalog.LuminosityMeasurement.filter">
              <IDREF>_2massK</IDREF>
            </REFERENCE>
            <ATTRIBUTE dmrole="sample:catalog.LuminosityMeasurement.value">
              <COLUMN dmtype="ivoa:RealQuantity" ref="_magK"/>
            </ATTRIBUTE>
            <ATTRIBUTE dmrole="sample:catalog.LuminosityMeasurement.error">
              <COLUMN dmtype="ivoa:RealQuantity" ref="_errK"/>
            </ATTRIBUTE>
          </INSTANCE>
          <INSTANCE dmtype="sample:catalog.LuminosityMeasurement">
            <ATTRIBUTE dmrole="sample:catalog.LuminosityMeasurement.type">
              <LITERAL value="magnitude" dmtype="sample:catalog.LuminosityType"/>
            </ATTRIBUTE>
            <REFERENCE dmrole="sample:catalog.LuminosityMeasurement.filter">
              <IDREF>_2massJ</IDREF>
            </REFERENCE>
            <ATTRIBUTE dmrole="sample:catalog.LuminosityMeasurement.error">
              <COLUMN dmtype="ivoa:RealQuantity" ref="_errJ"/>
            </ATTRIBUTE>
            <ATTRIBUTE dmrole="sample:catalog.LuminosityMeasurement.value">
              <COLUMN dmtype="ivoa:RealQuantity" ref="_magJ"/>
            </ATTRIBUTE>
          </INSTANCE>
          <EXTINSTANCES>SDSS_MAGS</EXTINSTANCES>
        </COMPOSITION>
      </INSTANCE>
    </TEMPLATES>
    <TEMPLATES tableref="_sdss_mags">
      <INSTANCE dmtype="sample:catalog.LuminosityMeasurement" ID="SDSS_MAGS">
        <CONTAINER>
          <FOREIGNKEY>
            <PKFIELD>
              <COLUMN dmtype="ivoa:string" ref="_container"/>
            </PKFIELD>
            <TARGETID>_source</TARGETID>
          </FOREIGNKEY>
        </CONTAINER>
        <REFERENCE dmrole="sample:catalog.LuminosityMeasurement.
            ↪ filter">
          <FOREIGNKEY>
            <PKFIELD>
              <COLUMN dmtype="ivoa:string" ref="_filter"/>
            </PKFIELD>
            <TARGETID>_SDSS_FILTERS</TARGETID>
          </FOREIGNKEY>
        </REFERENCE>
        <ATTRIBUTE dmrole="sample:catalog.LuminosityMeasurement.value
            ↪ ">
          <COLUMN dmtype="ivoa:RealQuantity" ref="_mag"/>
        </ATTRIBUTE>
        <ATTRIBUTE dmrole="sample:catalog.LuminosityMeasurement.error
            ↪ ">
          <COLUMN dmtype="ivoa:RealQuantity" ref="_eMag"/>
        </ATTRIBUTE>
      </INSTANCE>
    </TEMPLATES>
  </VODML>
  <RESOURCE ID="table_objects">
    <TABLE ID="_table1">
      <FIELD datatype="char" arraysize="16" ID="_designation"/>
      <FIELD datatype="float" arraysize="1" ID="_ra"/>
      <FIELD datatype="float" arraysize="1" ID="_dec"/>
      <FIELD datatype="float" arraysize="1" ID="_magH"/>
      <FIELD datatype="float" arraysize="1" ID="_errH"/>
      <FIELD datatype="float" arraysize="1" ID="_magK"/>
      <FIELD datatype="float" arraysize="1" ID="_errK"/>
      <FIELD datatype="float" arraysize="1" ID="_errJ"/>
      <FIELD datatype="float" arraysize="1" ID="_magJ"/>
      <DATA>
        <TABLEDATA>
          <TR>
            <TD>08115826-0205336</TD>
            <TD>122.992773</TD>
            <TD>-2.092676</TD>
            <TD>15.718</TD>
            <TD>0.112</TD>
            <TD>15.460</TD>
            <TD>0.212</TD>
            <TD>0.096</TD>
            <TD>16.273</TD>
          </TR>
          <TR>
            <TD>08115683-0205428</TD>
```

```
            <TD>122.986794</TD>
            <TD>-2.095231</TD>
            <TD>15.103</TD>
            <TD>0.077</TD>
            <TD>14.847</TD>
            <TD>0.127</TD>
            <TD>0.060</TD>
            <TD>15.860</TD>
          </TR>
          <TR>
            <TD>08120809-0206132</TD>
            <TD>123.033734</TD>
            <TD>-2.103671</TD>
            <TD>13.681</TD>
            <TD>0.027</TD>
            <TD>13.675</TD>
            <TD>0.048</TD>
            <TD>0.025</TD>
            <TD>14.161</TD>
          </TR>
        </TABLEDATA>
      </DATA>
    </TABLE>
    <TABLE ID="_sdss_mags">
      <FIELD datatype="char" arraysize="6" ID="_filter"/>
      <FIELD datatype="char" arraysize="16" ID="_container"/>
      <FIELD datatype="float" arraysize="1" ID="_mag"/>
      <FIELD datatype="float" arraysize="1" ID="_eMag"/>
      <DATA>
        <TABLEDATA>
          <TR>
            <TD>sdss:r</TD>
            <TD>08120809-0206132</TD>
            <TD>23.0</TD>
            <TD>0.03</TD>
          </TR>
          <TR>
            <TD>sdss:g</TD>
            <TD>08120809-0206132</TD>
            <TD>23.2</TD>
            <TD>0.04</TD>
          </TR>
        </TABLEDATA>
      </DATA>
    </TABLE>
  </RESOURCE>
</VOTABLE>
```

7 VO-DML XML schema [NORMATIVE]

In order to keep the VOTable standard as stable as possible and to better separate the concerns of the different documents, the elements used for mapping data model instances are defined in a standalone schema, that is then imported by the VOTable 1.4 schema itself.

7.1 Additions to VOTable

The new VOTable 1.4 VOTable schema (Ochsenbein 1900) simply imports the mapping schema through `xs:import`:

```
<xs:import namespace="http://www.ivoa.net/xml/VODML_Mapping/v1.0"
              schemaLocation="VODML-mapping.xsd"/>
```

And then defines a new element **`VODML`** as a direct child of **`VOTABLE`** (line 5):

```
 1    <!-- VOTable is the root element -->
 2    <xs:element name="VOTABLE">
 3        <xs:complexType>
 4            <xs:sequence>
 5                <xs:element name="VODML" type="vodml:VODML" minOccurs="0" />
 6                <xs:element name="DESCRIPTION" type="anyTEXT" minOccurs="0" />
 7                <xs:element name="DEFINITIONS" type="Definitions" minOccurs="0" />
 8                <!-- Deprecated -->
 9                <xs:choice minOccurs="0" maxOccurs="unbounded">
10                    <xs:element name="COOSYS" type="CoordinateSystem" />
11                    <!-- Deprecated in V1.2 -->
12                    <xs:element name="GROUP" type="Group" />
13                    <xs:element name="PARAM" type="Param" />
14                    <xs:element name="INFO" type="Info" minOccurs="0" maxOccurs="unbounded" />
15                </xs:choice>
16                <xs:element name="RESOURCE" type="Resource" minOccurs="1" maxOccurs="unbounded" />
17                <xs:element name="INFO" type="Info" minOccurs="0" maxOccurs="unbounded" />
18            </xs:sequence>
19            <xs:attribute name="ID" type="xs:ID" />
20            <xs:attribute name="version">
21                <xs:simpleType>
22                    <xs:restriction base="xs:NMTOKEN">
23                        <xs:enumeration value="1.3" />
24                    </xs:restriction>
```

```
            </xs:simpleType>
          </xs:attribute>
        </xs:complexType>
      </xs:element>
```

7.2 VODML-mapping shema

Listing 4: VODML additions to the VOTable schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    ↪ elementFormDefault="qualified"
  xmlns="http://www.ivoa.net/xml/VODML_Mapping/v1.0" targetNamespace
    ↪ ="http://www.ivoa.net/xml/VODML_Mapping/v1.0"
  >
  <!-- VODML types -->

  <xs:complexType name="VODML">
    <xs:annotation>
      <xs:documentation>
      A VODML element MUST have at least one model and at least on
        ↪ globals or templates.
      </xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="MODEL" type="Model" minOccurs="1" maxOccurs="
        ↪ unbounded" />
      <xs:choice minOccurs="1" maxOccurs="unbounded">
        <xs:element name="GLOBALS" type="VODMLGlobals"/>
        <xs:element name="TEMPLATES" type="VODMLInstanceTemplates"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="VODMLGlobals">
    <xs:annotation>
      <xs:documentation>
        This section will describe all global instances, that is,
          ↪ instances that are not created once per row of a table
          ↪ .
        It starts with a list of all the MODELs, then has all the
          ↪ GLOBAL instances.
      </xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="INSTANCE" type="VODMLObject" minOccurs="1"
        ↪ maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="ID" type="xs:ID" use="optional"/>
  </xs:complexType>

  <xs:complexType name="VODMLInstanceTemplates">
    <xs:annotation>
      <xs:documentation>
        This section will describe all the instance, that is,
            ↪ instances created once per row of a table.
        The instances can have FIELDrefs describing how to fill
            ↪ individual primtive valuesthe templates from TABLEDATA
            ↪  values.
      </xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="INSTANCE" type="VODMLObject" minOccurs="1"
          ↪ maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="tableref" type="xs:IDREF" use="required"/>
  </xs:complexType>

  <xs:complexType name="Model">
    <xs:sequence>
      <xs:element name="NAME">
        <xs:simpleType>
          <xs:restriction base="xs:string" > <!-- xsd:NCName ? -->
            <xs:pattern value="[a-zA-Z][a-zA-Z0-9_\-]*">
              <xs:annotation>
                <xs:documentation>
                  The name fof the model that is to be used as prefix
                      ↪ when referring to its elements in a VODMLRef.
                </xs:documentation>
              </xs:annotation>
            </xs:pattern>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="URL" type="xs:anyURI" />
      <xs:element name="IDENTIFIER" type="xs:string" minOccurs="0" >
      <xs:annotation>
      <xs:documentation>
      The IVOA Identifier by which the model is registered in an IVOA
          ↪  registry.
      </xs:documentation>
      </xs:annotation>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="ID" type="xs:ID" />
  </xs:complexType>

```

```xml
<xs:complexType name="VODMLInstance" abstract="true">
  <xs:attribute name="dmtype" type="VODMLRef" use="required" />
  <xs:attribute name="ID" type="xs:ID">
    <xs:annotation>
      <xs:documentation>
        Can be used by references as identifier for an object or
            ↪ template.
      </xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:complexType>

<xs:complexType name="VODMLObject">
  <xs:complexContent>
    <xs:extension base="VODMLInstance">
      <xs:sequence>
        <!-- lets be explicit -->
        <xs:element name="PRIMARYKEY" type="VODMLObjectIdentifier"
            ↪ minOccurs="0" >
          <xs:annotation>
            <xs:documentation>
              Objects, i.e. ObjectType instances, can have a unique
                  ↪ identifier which can be used in ORM-like
                  ↪ references.
            </xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="CONTAINER" type="VODMLReference" minOccurs
            ↪ ="0" maxOccurs="1" >
          <xs:annotation>
            <xs:documentation>
            Possible reference to a parent container of the object.
                ↪ May be given when the objects is not already
            contained in a colleciton on the parent object.
            Note, a VODMLReference can have multiple instances, but
                ↪ a CONTAINER MUST have only 1 instance.
            </xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="ATTRIBUTE" type="VODMLAttribute" minOccurs
            ↪ ="0" maxOccurs="unbounded" />
        <xs:element name="COMPOSITION" type="VODMLComposition"
            ↪ minOccurs="0" maxOccurs="unbounded" />
        <xs:element name="REFERENCE" type="VODMLReference" minOccurs
            ↪ ="0" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

```

<xs:complexType name="VODMLPrimitive"> <!-- in place of PARAM -->
 <xs:complexContent>
  <xs:extension base="VODMLInstance">
   <xs:sequence>
    <xs:element name="OPTIONMAPPING" type="VODMLOptionMapping"
      ↪ minOccurs="0" maxOccurs="unbounded">
     <xs:annotation>
      <xs:documentation>
      Allows one to map OPTION values in VOTABLE to either
        ↪ EnumLiterals in data model (if TYPE identifies VO
        ↪ -DML/ Enumeration),
      or SKOSConcept in external SKOS vocabulary (if ROLE is a
        ↪  VO-DML/Attribute containing a skosconcept
        ↪ declaration).
      NB: IF the datatype of the ROLE is an Enumeration, and
        ↪ there is NO optionmapping it implies that the
        ↪ values ARE the enum literals, or the concepts.
      </xs:documentation>
     </xs:annotation>
    </xs:element>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>


<xs:complexType name="VODMLLiteral"> <!-- in place of PARAM -->
 <xs:complexContent>
  <xs:extension base="VODMLPrimitive">
   <xs:attribute name="value" type="xs:string" use="required"/>
   <xs:attribute name="unit" type="xs:string" />
  </xs:extension>
 </xs:complexContent>
</xs:complexType>


<xs:complexType name="VODMLOptionMapping">
<xs:annotation>
 <xs:documentation>
 Allows one to map particular values defined in a VALUES/OPTION
   ↪ list to enumeration literals
 in the VO-DML model or to a concept in a SKOS vocabulary.
 </xs:documentation>
</xs:annotation>
 <xs:sequence>
  <xs:element name="OPTION" type="xs:string" minOccurs="1"
    ↪ maxOccurs="1" >
   <xs:annotation>
```

```
        <xs:documentation>
        The VOTable OPTION value that is being maped to enum literal or semantic concept.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:choice>
      <xs:element name="ENUMLITERAL" type="VODMLRef" minOccurs="1" maxOccurs="1" />
      <xs:element name="SEMANTICCONCEPT" type="xs:anyURI" minOccurs="1" maxOccurs="1" >
        <xs:annotation>
          <xs:documentation>
          TBD anyURI as an identifier of concepts made sense for SKOS vocabularies. How about general semantic vocabularies?
          I.e. is it ok for the type be xs:string iso xs:anyURI?
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:choice>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="VODMLFieldOrParamRef">
  <xs:complexContent>
    <xs:extension base="VODMLPrimitive">
      <xs:attribute name="ref" type="xs:IDREF" use="required" />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>


<xs:complexType name="VODMLRole">
    <xs:attribute name="dmrole" type="VODMLRef" use="optional">
      <xs:annotation>
        <xs:documentation>
        NB: ROLE has minOccurs=0 , at the moment only because VODMLObject::CONTAINER (a VODMLReferece) needs no role.
        ATTRIBUTE, COMPOSITION and REFERENCE MUST have a ROLE.
        Hard to model in XML schema, could be done in Schematron.
        </xs:documentation>
      </xs:annotation>
    </xs:attribute>
</xs:complexType>

<xs:complexType name="VODMLAttribute">
  <xs:complexContent>
  <xs:extension base="VODMLRole">
    <xs:choice>
     <xs:choice maxOccurs="unbounded">
      <xs:element name="COLUMN" type="VODMLFieldOrParamRef">
       <xs:annotation>
        <xs:documentation>
         When used inside a "template" structured type, i.e.
           ↪  one defined inside a TABLE element,
         this allows one to indicate a FIELD representing
           ↪ the attribute.
        </xs:documentation>
       </xs:annotation>
      </xs:element>
      <xs:element name="CONSTANT" type="VODMLFieldOrParamRef">
       <xs:annotation>
        <xs:documentation>
         Ref to a predefined PARAM.
        </xs:documentation>
       </xs:annotation>
      </xs:element>
      <xs:element name="LITERAL" type="VODMLLiteral">
       <xs:annotation>
        <xs:documentation>
         Simple, primitive value, possibly with extra
           ↪ attributes.
         Similar to PARAM, but restricted attribute set.
        </xs:documentation>
       </xs:annotation>
      </xs:element>
     </xs:choice>
     <xs:element name="INSTANCE" type="VODMLObject" maxOccurs="
       ↪ unbounded">
      <xs:annotation>
       <xs:documentation>
        Structured value, must be instance of DataType
       </xs:documentation>
      </xs:annotation>
     </xs:element>
    </xs:choice>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="VODMLComposition">
 <xs:annotation>
  <xs:documentation>
  A VODMLCollection represents collection of child objects in a
    ↪ VO-DML Composition relationship.
    The collection receives the VO-DML ref to the composition
        ↪ relation, the member objects inside the
    collection do *not* have a ROLE.
    </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="VODMLRole">
      <xs:sequence>
        <xs:element name="INSTANCE" type="VODMLObject" maxOccurs="
            ↪ unbounded" minOccurs="0">
          <xs:annotation>
            <xs:documentation>
            An object in the collection. It type must conform to the
                ↪  declared type of the VO-DML Collection.
            I.e. it must be that exact type or a sub-type.
            </xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="EXTINSTANCES" type="xs:IDREF" maxOccurs="
            ↪ unbounded" minOccurs="0">
          <xs:annotation>
            <xs:documentation>
           Reference to a VODMLOBJECT declaration possibly
               ↪ containing child objects for this composition
               ↪ relation.
            </xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>




<xs:complexType name="VODMLReference">
  <xs:annotation>
    <xs:documentation>
      Provides a reference to an ObjectType instance. Must allow
          ↪ that instance to be identified exactly.
      VArious different modes depending on how that instance is
          ↪ serialized.
      - If as a Standalone instance in same VOTable document, an
          ↪ IDREF can point to its ID (use IDREF).
      - If as a row in a TABLE a relational foreign key can be used
          ↪  (use ORMREFERENCE)
      - if a remote document contains the serialized instance, a
          ↪ URI must be used that MUST be able to identify that
```

```
              ↪ object
          inside its remote serialization (a REMOTEREFERENCE must be
              ↪ used).
        </xs:documentation>
      </xs:annotation>
      <xs:complexContent>
        <xs:extension base="VODMLRole">
          <xs:sequence>
            <xs:choice maxOccurs="unbounded">
              <xs:element name="IDREF" type="xs:IDREF">
                <xs:annotation>
                  <xs:documentation>
                    MUST identify an individual/standalone object defined
                        ↪  in the same XML document.
                  </xs:documentation>
                </xs:annotation>
              </xs:element>
              <xs:element name="REMOTEREFERENCE" type="xs:anyURI" />
              <xs:element name="FOREIGNKEY" type="VODMLORMReference" />
            </xs:choice>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>

    <xs:complexType name="VODMLORMReference">
      <xs:annotation>
        <xs:documentation>
          A reference to an object identified by that object's
              ↪ identifier.
          The referenced object must be stored in a TABLE and must have
              ↪  been annotated with an explicit identifier.
        </xs:documentation>
      </xs:annotation>
      <xs:complexContent>
        <xs:extension base="VODMLObjectIdentifier">
          <xs:sequence>
            <xs:element name="TARGETID" type="xs:IDREF" minOccurs="0">
              <xs:annotation>
                <xs:documentation>
                  This element MAY be used to provide a IDREF to the (ID
                      ↪  of a) VODMLObject template annotating
                  the TABLE containing the referenced object.
                </xs:documentation>
              </xs:annotation>
            </xs:element>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
```

```
</xs:complexType>

<xs:complexType name="VODMLObjectIdentifierField">
  <xs:choice>
    <xs:element name="LITERAL" type="VODMLLiteral"/>
    <xs:element name="COLUMN" type="VODMLFieldOrParamRef"/>
    <xs:element name="CONSTANT" type="VODMLFieldOrParamRef"/>
  </xs:choice>
</xs:complexType>

<xs:complexType name="VODMLObjectIdentifier">
  <xs:annotation>
    <xs:documentation>
      This type allows a generic identifier to be assigned to an
          ↪ object. The identifier consists of one or more IDFIELD
          ↪ -s.
      This way of identifying an object is equivalent to using one
          ↪ or more columns in a table as primary key.
    </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="PKFIELD" maxOccurs="unbounded" type="
        ↪ VODMLObjectIdentifierField">
      <xs:annotation>
        <xs:documentation>
          A field in an identifier. The identifier may contain 1 or
              ↪ more such fields. Their order
          is important, ORM references to the object must use the
              ↪ same order for their foreign key.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:simpleType name="VODMLRef">
  <xs:annotation>
    <xs:documentation>
      The valid format of a reference to a VO-DML element. (Used to
          ↪  be 'UTYPE').
      MUST have a prefix that elsewhere in the VOTable is defined
          ↪ to correspond to a VO-DML model defining the
          ↪ referenced
      element.
      See "mapping document", https://volute.g-vo.org/svn/trunk/
          ↪ projects/dm/vo-dml/doc/MappingDMtoVOTable-v1.0-201607
          ↪ xx.docx.
      Suffix, separated from the prefix by a ':', MUST correspond
          ↪ to the vodml-id of the referenced element in the
      VO-DML/XML representation
      of that model.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-zA-Z][a-zA-Z0-9_\-]*:[a-zA-Z][a-zA-Z0-9\.
    ↪ _]*" />
  </xs:restriction>
 </xs:simpleType>

</xs:schema>
```

References

Graham, Matthew. 2013. “UTypes: Current Usages and Practices in the IVOA.” http://www.ivoa.net/documents/Notes/UTypesUsage/index.html.

Lemson, Gerard. 2015. “VO - DML: A Consistent Modeling Language for IVOA Data Models.” http://www.ivoa.net/documents/VODML/20151010/index.html.

Ochsenbein, Francois. 1900. “VOTable Format Definition.” http://www.ivoa.net/documents/xxxx.

“PR on Use of SKOS Vocabularies in IVOA.” 1900.

“UCDs.” 1900.