VO-DML and UTYPEs and VOTable –Use Cases

Author: Gerard Lemson

Date: 2012-02-12

Version: 0.1

Table of Contents

Conclusions

In this document we investigate the use cases on the UTYPE Tiger Team wiki in the light of the proposal in ...

Conclusions are:

- If we want to support annotations of *generic* relational database we either have to add GROUP-ing to TAP_SCHEMA, or allow a VOTable with TABLEs-without-DATA to annotate them. If we only want to support serialization according to prescribed OR-mapping approach, utype annotation is strictly speaking not required, but an ORM strategy must be agreed on. Could still include explicit annotation also supported for generic databases.
- DataLink may be required to annotate FITS files or other generic tabular data formats.
- We may not be able to treat every type of transformation n a simple manner. For example, if users break links between object, one cannot deserialize instances properly.
- Many of the issues relate to the impedance mismatch between an OO model and a relational database. OR mapping strategies have learned to work with this. We should try to follow best practices as much as possible. Example, JPA. How does it allow annotation of classes. Basically using Grouping. For the Java class IS mapped to a type with multiple columns.
- Some use cases are specified in terms of the, as yet, ill defined concept of "an instance of a data model". Meant is probably some "standard" representation of instances. The XMl schema in vo-dml-instance.xsd contains a proposal for a standard serialization format that is maximally faithful to the VO-DML/XML representation of the data model. Now some of the use cases can be rewritten using an assumed document following this standard serialization as either target or source. Then in some cases one can try to write solutions as XSLT scripts, rather than relying on code. For example an XSLT script could be provided that could turn a VOTable into this standard serialization. The other way around would always need to assume a default VOTable representation of the instance.
-

1 UC #1

Serialize DM instances to file: given an instance of a Data Model and the DM machine readable description, a writer can serialize the instance into a number of supported tabular formats. The writer could be a DAL service.

If a generic table format is the target, we must assume an ORM-like strategy, i.e. we cannot use GROUPs for example to store instances directly. We must be allowed to assume multiple tables can be stored in the file if necessary for representing the model. VO-URP has shown how to automate an OR-mapping for data models defined using the VO-URP "intermediate model", the direct ancestor of

VO-DML. VO-URP also shows how to map the data model to Java and XML schema and how to transform between the different representations.

So it is possible to map a data model to a relational database, from which a simple dump of all tables to the tabular format would enact the required serialization. These tables can be annotated with corresponding tables-without-DATA in an external VOTable (associated to it via DataLink spec??) that provides all metadata for the tables and in particular can use its internal GROUP-s to represent attributes with structured datatypes.

Such an annotated VOTable exporter must still be written but can be generated using an XSLT script similar to the one that produced the TAP_SCHEMA and DDLs in VO-URP.

2 UC #2

Deserialize DM instance from file: given a serialized instance of a Data Model in a supported tabular format and the DM machine readable description, a reader can deserialize the instance into memory, building an object consistent with the DM itself.

This is the goal of the mapping proposal and should probably be the reference implementation of our effort. For if we can show to do this then any client tool MIGHT use the reference implementation.

Note that the data model instance can only be completely built, if the serialization contains all information.

VO-URP shows how this can be done from a relational database, using JPA annotation on Java classes generated from the data model.

Here we can try to mimic this based on a VOTable. A VOTable interpreter must be written that can use annotation to infer what types are stored, create instances of these using some factory method, fill them with data using the existing setters based on property names. A Hashtable of these objects keyed by the declared identifiers can be used to resolve references.

3 UC #3

Trivial round-tripping: given a serialized instance of a Data Model in a supported tabular format, an I/O library (possibly model-unaware) can convert the instance into a different, supported format without breaking its VO compliance.

VO-URP shows how this can be done without requiring utypes using the forerunner of VO-DML.

Using a VOTable-interpreter that can instantiate the VO-URP Java instances from an annotated VOTable the same can be done for that format.

4 UC #4

Represent an arbitrary number of instances of the same class in a DM instance (for example, N instances of the PhotometryFilter class in a PhotometryCatalog instance of the Spectral DM).

[Omar: in UTypes terms this means that the same UType could be used several times to describe attributes of several different instances, in the same file. Also, several Utyped values should be bundled together in some way, so that each instance of the class can be reconstructed].

See the examples in the main document. There multiple LuminosityMeasurement instances all related to a common Source are stored in various ways.

4.1 UC #4a

VO-Importer: given a non-compliant file (or set of files) and the library of all the DM descriptions, an importer application can allow users to map columns and parameters in the file (or the set of files, or database) to IVOA DM attributes, thus producing a compliant version of the file (or set of files).

TBD

5 UC #5

VO-Publisher: given a database and the library of all the DM descriptions, a helper application can allow data providers to map tables and columns in a database to IVOA DM attributes, in order to build a DAL service.

TBD

I have some doubts about this. If the idea is that one should be able to have some tool to only identify tables to single types, columns to attributes, this is not sufficient to map generic representations. What such a tool might achieve and how complex it needs to be to do so depends completely on how the database has actually serialized the model. As example consider the database to be the result of running one or more SQL queries against a database designed according to the standard mapping. Say the query is

The simple point-and-click method for identifying columns with attributes is not sufficient. For example one must be able to GROUP columns together and one must be able to identify foreign keys derived from different relationships. Maybe a tool can be written to assist in this, and business would be very interested in it I think, as it would solve a lot of data-warehouse-construction problems.

6 UC #6

VO-Query: given a compliant archive/service it is possible to query it by using Utypes to refer to data model elements (for instance, query all observations for a target whose name is given). [Omar: I am not sure I understand this: does it mean that I could, for instance, query an archive for all the SDSS.g and SDSS.u magnitudes using Utypes?]

I assume the archive is relational and the tables have been annotated with a VOTable-without-DATA. One can query the VOTable first using XSLT/XPath for the TABLE.COLUMN combination that corresponds to the data model element, and use that to construct my SQL query.

7 UC #7

Data Model representation: DMs should be represented by a machine readable description that allows to:

VO-DML provides this.

7.1 UC #7.0

Mapping from one simple piece of metadata to one single data model attribute at the finest description level

TBD

7.2 UC #7.1

Describe and document Data Model elements.

VO-DML includes description elements. A simple HTML script is available to serialize the model to document.

7.3 UC #7.2

Keep versioning information about the DM.

VO-DML contains a Model type that has a field for the version label and a URI pointing to a previous version.

7.4 UC #7.3

Reuse an existing DM in a new DM.

VO-DML allows one to import another data model. Types in the other model can be used as datatype of properties, or as base classes, by referring to them by their utype. A utyperef attribute takes care of this. A Schemtron file exists that can check correctness of the usage.

7.5 UC #7.4

Extend an existing DM with a new DM.

Same as UC#7.3

7.6 UC #7.5

Abstract the creation of VO-compliant I/O libraries from the details of the single DM. According to the programming language, each DM would be represented by some kind of plugin of the generic library. [Omar: this use case is actually a consequence of the implementation of the other 7.x cases].

Don't see why this UC is a consequence of others. Asking for a generic software library seems not implied by other cases.

But VO-URP shows how this can be done. We have a generic framework of base classes that is used by Java classes generated for a particular model and allows them to be handled, references to be resolved, (un)marshaling vs XML, storage in a database using JPA.

We do not yet have an implementation for model reuse, but that should be straightforward.

8 UC #8.

Link columns in a relational model of the registry to VOResource schema elements

TBD

If a VO-DML version of VORegistry were to exist, utype-s would be generated. If a "standard" OR-mapping is performed as described in UC-s above, the relational model of the registry is automatic.

If a custom mapping to a relational model is performed, this mapping must be custom annotated. D

9 Concrete Use Cases

9.1 (Photometry Catalog, points in columns)

Represent a Photometry Catalog with a definite number of Magnitudes expressed in columns and astronomical sources in rows. For example, an SDSS catalog with the following columns:

SDSSID | RA | DEC | U | G | R | I | Z

This seems to me a question of constructing the right data model rather than being a constraint on utype-s. The Sample model supports this already and examples have been given that provide this kind of flattened mapping with proper annotation.

9.2 (Photometry Catalog, points in rows)

A Photometry Catalog could refer to a single object observed in a number of filters, or to different objects observed in a number of filters, and the filters could be an arbitrary number. Employing an efficient relational approach would suggest to represent this as a table where each magnitude is expressed in a different row, and the other information (object name, coordinates, instrument, filter, etc) are in columns, or are factored out in the table header if they are common to all points.

For instance, here is a (simple) example of an (unnormalized) catalog for different sources. Notice that this table doesn't use any controlled vocabulary for filters, target names and instruments, while VO documents should:

TargetName | RA | DEC | Instrument | Filter | Magnitude | Units

M51 | xx.yy | xx.yy | SDSS | u | xx | ABMAG

M51 | xx.yy | xx.yy | SDSS | g | xx | ABMAG

NGC1068 | xx.yy | xx.yy | GALEX | nuv | Jy

This seems to me a question of constructing the right data model rather than being a constraint on utype-s. The Sample model supports this already and examples have been given that provide this kind of OR-like mapping of parent-collection pattern with proper annotation.

9.3 (Aggregated SED)

An Aggregated SED is defined as an aggregation of different segments of spectro-photometric data, where each segment can be a Photometry Catalog, a Spectrum or an entire SED itself. It should be possible to serialize an SED as a list of several tables, each table representing a segment. Complex formats like VOTable and FITS can allow the tables to be stored in the same file.

This seems to me a question of constructing the right data model rather than being a constraint on utype-s. The Sample model has individual magnitudes, if a model for Segment were added, possibly with a collection of Pixel-s (wavelength, bin, flux, error,..) than an ordinary OR mapping will do the trick. Examples have been given that provide this kind of flattened mapping with proper annotation.

9.4 (STC serialization)

It has to be possible to embed STC instances in tables and attach these instances to other objects in the table. This is a generic example of Model reuse (the STC model is reused by other models)

The main issue is that STC does not obey many rules of data modeling. It is an XML schema "gone wild". A normalization of this model is required to be able to interpret the model interms of VO-DML concepts, if not a full VO-DML description. This is LONG overdue, though admittedly a non-trivial task.

One might first start with trying to refactor the schema according to design rules accepted by both Registry and VOTable: no <element ref="">, no substitutiongroup, no attributelist etc.

9.5 (STC serialization in a VOTable)

Since right now, the only non-deprecated way to include STC metadata in VOTables relies on utypes, it is particularly unfortunate we have no way of doing this that's REC. I'm counting this as a separate use case since IMHO it's a particular shame something as basic is almost undefined in our recommended format -- the format we, as the VO community, actually control.

Also, IMHO ideally clients should not have to worry about what (if any) data model some data within a VOTable conforms to. Just as any VOTable library could support (the now deprecated) COOSYS element, support for "modern" STC metadata should be "generic".

Sorry for littering the use case with discussions on practice; if you have a better place for this stuff, please do move it there.

One plan to do this is described in Referencing STC in VOTable. The basic idea is to collect all information pertaining to STC (or even some other data model) in one group, like this:

```
<GROUP utype="stc:CatalogEntryLocation">
  <PARAM name="CoordFlavor" datatype="char" arraysize="*"
    utype="stc:AstroCoordSystem.SpaceFrame.CoordFlavor"
    value="SPHERICAL"/>
  <PARAM name="CoordRefFrame" datatype="char" arraysize="*"
    utype="stc:AstroCoordSystem.SpaceFrame.CoordRefFrame"
    value="ICRS"/>
  <PARAM name="ReferencePosition" datatype="char" arraysize="*"
    utype="stc:AstroCoordSystem.TimeFrame.ReferencePosition"
    value="BARYCENTER"/>
  <PARAM name="TimeScale" datatype="char" arraysize="*"
    utype="stc:AstroCoordSystem.TimeFrame.TimeScale" value="TT"/>
  <PARAM name="Epoch" datatype="char" arraysize="*"
    utype="stc:AstroCoords.Position2D.Epoch" value="2010.2"/>
  <PARAM name="yearDef" datatype="char" arraysize="*"
    utype="stc:AstroCoords.Position2D.Epoch.yearDef" value="J"/>
  <PARAM name="TimeInstant" datatype="char" arraysize="*"
    utype="stc:AstroCoords.Time.TimeInstant"
    value="2002-01-28T09:30:00"/>
  <PARAM name="URI" datatype="char" arraysize="*"
    utype="stc:DataModel.URI"
    value="http://www.ivoa.net/xml/STC/stc-v1.30.xsd"/>
  <FIELDref ref="raErr"
    utype="stc:AstroCoords.Position2D.Error2.C1"/>
  <FIELDref ref="deErr"
    utype="stc:AstroCoords.Position2D.Error2.C2"/>
  <FIELDref ref="ra" utype="stc:AstroCoords.Position2D.Value2.C1"/>
  <FIELDref ref="de" utype="stc:AstroCoords.Position2D.Value2.C2"/>
  <FIELDref ref="pmra"
    utype="stc:AstroCoords.Velocity2D.Value2.C1"/>
  <FIELDref ref="pmde"
    utype="stc:AstroCoords.Velocity2D.Value2.C2"/>
</GROUP>
<FIELD ID="ra" name="ra" datatype="float"/>
<FIELD ID="de" name="de" datatype="float"/>
<FIELD ID="raErr" name="raErr" datatype="float"/>
<FIELD ID="deErr" name="deErr" datatype="float"/>
<FIELD ID="pmra" name="pmra" datatype="float"/>
<FIELD ID="pmde" name="pmde" datatype="float"/>
```

I think to translate this use case to the approach using VO-DML+mapping requires a sensible model of those parts of STC one may wish to have in a VOTable. Then one might follow the "VO-DML mapping" approach. Maybe this should be considered a use case in the refactoring of STC into a more sensible design. See also 9.6.

Alternative is to grandfather STC together with other existing model-UTYPE-list combinations, treat them as special cases in VOTable annotation.

Btw, I think the annotation in the VOTable example is a denormalization of the real model, i.e. not a direct representation of it.

9.6 (STC in a Spectrum)

This is a more specific example of model reuse. Spectral DM uses some parts of the STC DM to describe the reference frame of the observations: the frame can include several axes: time, spectral coordinate, flux, photometry filters. Some of these axes could be different instance of the same STC or STC-derived class, like the photometry filters. This has to be represented in a generic tabular format using Utypes to describe the structure of the serialized instances. Different instances of the same class must somehow be disentangled from the others.

Nice exercise to try to write the part of STC used by Spectrum as the start of a refactoring of STC.

9.7 (Create compliant SSA service using a database and a non-compliant archive)

Given a database and an archive containing spectra serialized in a non-compliant way (but in a supported format, like FITS), a data publisher might want to create a VO service: in principle this would require the creation of a new database (or a view on the original one) and to copy and

change the headers of the non-compliant files. A more efficient solution would be to leave the archive and the database untouched and to add an additional layer on top of them: the layer would add the required metadata to the original files on the fly (see R #4). For example, the service can read the information in the database and fill a VOTable compliant header (putting together the database values with the predefined Utypes) that will wrap the original FITS file in the files response.

This seems mainly a coding exercise. VO-URP shows it is possible to serialize a VO-DML data model into Java classes. Code could be written that creates instances of these from data read form the non-standard archive. This code implements the mapping of the archive to the data model. The Java instances can be marshaled into standard XML, or transformed further to the format required by SSA.