International
Virtual
Observatory
Alliance

Utypes: a standard for serializing Data models instances
Version 0.7

IVOA Working Draft May 23, 2012

This version:

http://www.ivoa.net/Documents/Notes/WD-Utypes-0.7-20120523.pdf

Latest version:

http://www.ivoa.net/Documents/latest/Utypes.html

Previous versions:

0.6

Editor(s):

Mireille Louys, Omar Laurino

Authors:

Mireille Louys, Omar Laurino, Laurent Michel, Doug Tody, Markus Demleitner, François Bonnarel, Alberto Micol, Gerard Lemson, Mark Cresitello-Dittmar, Jonathan McDowell, etc.

Abstract

This document discusses the definition, usage, interpretation and implementation of Utypes in the Virtual Observatory. It defines standard serialization strategies for representing astronomical datasets according to IVOA standards in various file formats. Data model extensions using customly designed objects are also considered.

Status of this document

This document has been produced by the Data Model Working Group. It is still a draft.

Acknowledgements

Members of the IVOA Data Model Working Group, including representatives of the US VAO, the Euro-VO, GAVO, VO-France, etc. have contributed to the present draft.

Contents

1 Introduction

1.1 Scope of the document

This document is summarizing the practice adopted in the Virtual Observatory for naming and identifying data models elements. It formulates use-cases for data model serialisation using data model items representation in text lists or tables. It defines the Utype concept, the syntax proposed to represent and publish Utypes-lists in the VO, and finally illustrates how to use them, in protocols and VO-aware applications.

1.2 Context

Interoperability in the virtual observatory requires to circulate data products and their metadata via protocols, so that they can be interpreted and used in VO-aware applications. The IVOA has elaborated data models to represent, in a logical framework, most kinds of metadata describing the content of data products in astronomy. We shall recall now how DM are produced in the IVOA and which representation can be used for circulating metadata across VO-tools.

1.2.1 Data modeling and its usage in the IVOA

The goal of data models defined in the IVOA project is to express the set of necessary concepts and their representation for describing astronomical data sets. The approach to describe metadata is based on object oriented programming. A UML representation of the model, namely the class diagram allows to show the main concepts as classes, their properties as attributes, relationships from one concept to another and dependencies. IVOA data models cover various types of astronomical data sets and are currently described using :

1. a graphical view as one or more UML class diagram
2. the text description of all data model items consisting in the main core of the IVOA standard document.
3. a hierarchical view as an XML schema
4. examples of data sets and their metadata description as XML, VOTable or FITS instance documents.

1.2.2 Serialisation

Serialisation is a process that helps to represent collections of metadata in a transportable way -that is outside programs, and in compatibility with

an IVOA Data Model. Models are designed in UML (Unified Modeling Language), using mainly the class diagrams. From these classes descriptions, the developer can derive a library in Java or C++ or Python, that can operate on these classes, and re-use them for his/her own application. If we need to save the content of a set of classes instances in a program , we 'll use one of the familiar VO file format: VOTable, FITS, XML or structured ASCII files. Flat formats In the flat formats , Utypes can be used in order to associate fields or elements in these files to data model items.

More precisely, there are 4 ways of using/exporting the data model structural organisation :

- implement the data model classes in an object oriented language. Then metadata associated with a 2D image for instance, are described by a set of classes within the Observation DM. To publish the metadata values of such an image, one just need to instantiate the corresponding classes of the data model and use the setters and getters functions of these classes to load or export the attribute values.
- derive an XML schema from the data model UML class diagram. Every class and every class's attribute will be translated as an element. Nested classes will produce a tree like structure in XML schema. The metadata for a specific dataset, for instance a spectrum of Vega, is then an XML instance document following the XML schema structure.
- define Utypes as data model labels that point to their associated data model element. Include them as tags in one of this file format:
  - a (keyword,value) list of metadata, for instance in ASCII
  - a text structured format like VOTable [7] or JSON [2] allowing for a nesting strategy to bring back the hierarchy.

Every serialisation format has its advantages:

- XML provides a hierarchy of nested objects and can directly use XML searching tools like XPath.
- The ASCII list of (keyword,value) pairs is the simplest most compact representation.
- VOTable encodes object nesting within GROUP elements and supports large collections of similar objects.

Hence all three should co-exist within the VO. The translation from one representation to the other should be bijective which implies that the Utype string must encode the nesting structure of the objects in the model.
Whereas the graph structure of the UML class diagram is richer than the

XML tree projection, the translation can still be organized both ways, provided some rules are adopted for the UML design as explained in Section 5.

1.3 Goals

Data models organise the metadata in classes and offer a logical way to understand and interpret their specific roles. It synthetises knowledge about the way the data were obtained, can be used, combined, accessed, etc. The design of classes and their relationships implicitely encodes some prioritisation on the metadata, driven by the use-cases the data models are built for. This is the reason why the detailed documentation of data model in IVOA standards is important in order to apply and use a data model in the best conditions.

When serializing complex, structured objects in tabular format, the structure and logical binding get lost in a flattened, unstructured set of table header definitions and cells. File readers don't have any means of reconstructing the original structure, unless they are provided with additional metadata that describes how the different columns and header parameters are meant to be bundled together.

This document describes how to serialize complex metadata structures in tabular formats or lists, so that the structure can be reconstructed by the reader and re-interpreted in a data model framework during deserialization, despite the flat nature of a table representation (or list). It explains the relationship between the various types of serialisation of data model instances using examples.

2 Requirements for interoperable serialisations

Here is a list of requirements for serialisation of data models in the VO.

1. A standard serialization strategy must allow clients to build a hierarchical representation of a dataset stored in a tabular format, according to an arbitrary data model, which is assumed to be unknown to the reader.
2. A standard serialization must refer to the data model it relies on and allow pointing to its documentation in an automatic fashion but also interactively.
3. A standard serialization strategy must allow clients to identify and extract objects from datasets stored in a tabular format, or a list, ac-

cording to a data model that is assumed to be in the client's conceptual domain.
4. The standard serialization strategy adopted to meet the previous requirements must be independent from the particular file format and the particular data models.

3 Use Cases

By meeting the aforementioned requirements, the serialization strategy described in this document enables, at least, the following use cases:

1. A user loads a VO-compliant dataset stored in a tabular format using any VO-enabled application and is presented with a structured, browseable representation of the elements stored in the dataset. This use case is independent from the particular data model represented in the file. This corresponds to a portal or a standard display tool, like TopCat for instance, where for each data in the table the application can pull-up a brief description of a field and show it to the user.
2. A user loads a VO-compliant file stored in a tabular format and coming from any VO-compliant service or application into a VO-enabled application tailored for a specific set of science cases. The reading application will find the list of business objects represented in the dataset and use them to provide the user with the relevant data, according to the application specific use cases. As an example, a service getting observation datasets compatible to ObsCoreDM, extracts the footprint information and displays it on an image . Note that the original VO-compliant file might have been originated from **any** VO-compliant service or application, not necessarily a service or application specifically tailored to provide files directly readable by the other application.[1]
3. A user can ask a VO-enabled application A, tailored for a specific set of science cases, to save a file using a tabular representation and according to a data model specific to the use case. The user then loads the file using a VO-enabled application B, tailored for a different set of use cases: the user is thus presented by B with the information relevant to its specific use cases.[2]

[1]For example, the original file contained photometry obtained from a TAP query and the client application is an SED builder. Also, neither application is aware of each each other, (*but they support a common set of metadata each of them can recognise.* Mireille

[2]For example, the user builds a SED in a SED builder, beams it to a fitting application using SAMP and the fitting application presents him with a dialog box that asks him to select which axes to fit and with which models. Note that the fitting program automatically

4 Utype definition

The UML class diagram is navigable, therefore any data model part is reachable from a main data model element using a logical path through classes and attributes. This is stored as a string and called a "**Utype**". This is uniquely defined for each piece of metadata described in the model and works as a semantic type, underlining the role of this piece of metadata with respect to the defined classes and attributes in a specific model. Such a path in the data model structure needs to be fully consistent to the data model piece it points to. This is illustrated in Fig. 1.

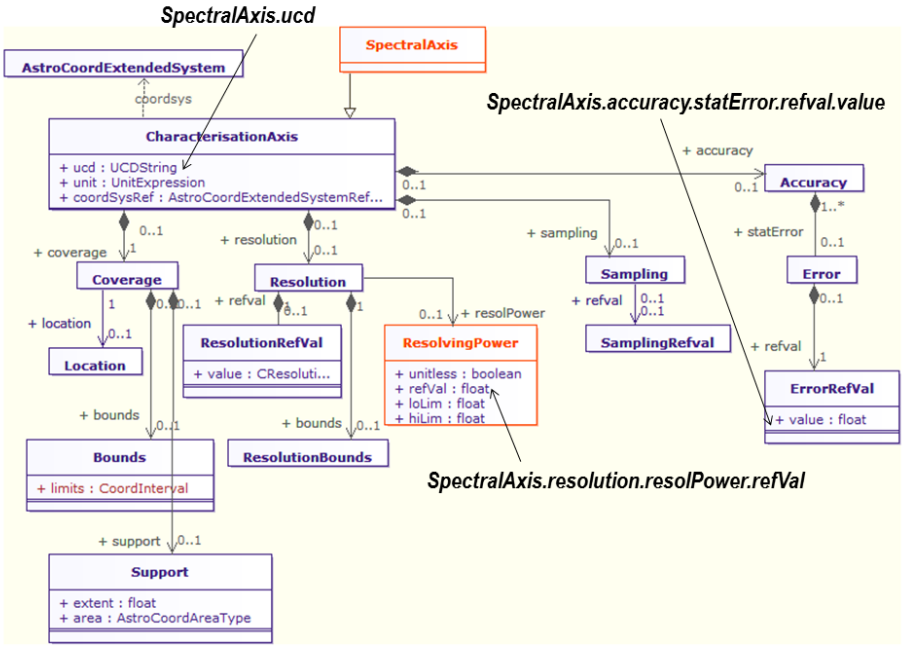

Figure 1: *Binding Utype labels to data model items: Excerpt from the Observation Core Components DM : Utypes work as path from major elements (here SpectralAxis) into the UML class diagram, designed as an acyclic graph.*

Therefore any IVOA data model item can be used by applications and protocols, if we provide for each:

identifies the data axes and their metadata (units, errors), homogenize them, for example converting the units to the same one, and then presents the user with a choice among the only relevant information: the axis names. If the application is specifically tailored for astronomical datasets, it might even recognize spectral axis and flux axis and simplify the user choices.

- a data type
- a utype
- a unit (explicit or implicit following the data model field definition)
- a mandatory status for this item (mandatory or optional)

Here are two examples:
In the SpectralDM, we identify a **Target** with its *name* and must specify the *data type* as "string", *utype* as "Target.Name", *unit* as "unitless", *status* as "optional". In ObsTAP, the data model item describing how a data product is calibrated is described in the data model summary as a table row like :

name	utype	units	UCD	data type	description	status
`calib_level`	Obs.calibLevel	unitless	meta.code; obs.calib	enum int 0,1,2,3	Calibration level of the observation:	MAN

Table 1: Calibration level as defined in the ObsCore data model

Other semantic tags, like UCDs already exist to classify metadata, they can categorise physical quantities but are not precise enough to uniquely identify a piece in a data model. As long as new data collections appear with many different metadata organisation, the need to bind one piece of metadata with its corresponding representation in an IVOA Data model (e.g. in SpectrumDM) is crucial to promote interoperability and make protocols and applications easier for the user.

4.1 Utype string properties

The Utype purpose is essentially to point to the simplest atoms of a data model, i.e attributes within a class, so that it can be used in any serialisation document to parametrize metadata values.

4.1.1 Utype string built as a path to a data model element

Composing a name for atomic elements is just using a string composition in most object programming language. For example, in Characterisation DM, pointing to the number of bins along the Spectral axis will be **SpectralAxis.numbins**. Most of attributes in data models are themselves classes, that can be browsed down in order to reach the lowest level of encapsulation and point to single value elements. This is the case for **SpectralAxis.coverage.location.unit** with 2 levels of nesting.

If a data model gets more complex, like for the Simulation DM ([5]), groups of classes involved in the same use-cases (functionalities) are identified and organised in packages. See the Simulation DM overview at :

http://www.ivoa.net/cgi-bin/twiki/bin/view/IVOA/IVOATheorySimDBDM. This should also be reflected in the string structure of Utypes.

Because of the graphical structure of UML, linking classes with each other, a path inside the data model is not always unique. In order to be able to build-up Utypes names directly from a UML data model, we propose the following rules for UML design, and developed a Utype syntax from it.

4.1.2 Restriction on the UML design

In order to facilitate translation to XML, the UML class diagram need to be acyclic , then to avoid too many associations between classes and exclude loops. Templates constructs are not supported. Collections of data models elements and references are allowed. Some rules have been designed during the Simulation data model elaboration and available in the VO-URP project. [**?**].(insert link for VO-URP)

4.1.3 Link any utype to the corresponding element definition in the original data model

The Utype string should be clearly related to its data model therefore any Utype string should begin with the data model name as a prefix, like for instance stc:AstroCoordSystem or obs:Observation.dataproductype. This prefix string can be interpreted as a data model name abbreviation. As defined in the StandardRegExt specification, we could extend the mechanism to register data model definitions in the IVOA registries. This would imply that:

- a data model can be registred
- the prefix can be interpreted to point to the data model registry entry
- a uri can be generated from the utype string to point to the related data model item documentation part.

Here is an example of what could be designed for instance for the Photometry data model or any other.

Examples of serialisation in various formats can be gathered in a directory as instance documents

This example suggests that all related documents are gathered under a specified directory and fully accessible, either for humans for data model browsing and interpretation or for machines to reuse datamodel components in applications.

The utype list in simple 'tsv' or other simple text format can be used by applications for syntax checking or utype validation. The URI mechanism

datamodel.name	phot
datamodel.uri	http://www.ivoa.net/Documents/PHOTDM
datamodel.std	http://www.ivoa.net/Documents/PHOTDM/REC-PhotDM-1.0-2012xx.pdf
datamodel.doc	http://www.ivoa.net/Documents/PHOTDM/PhotDM-1.0.html
datamodel.version	1.0
datamodel.xsd	PhotDM-v1.0.xsd
datamodel.utypelist	PhotDM-v1.0.utype.tsv
datamodel.examples	http://www.ivoa.net/Documents/PHOTDM/examples

Table 2: Gathering information in a registry entry or the like

described by N.Gray can be used to point to the dedicated section of documentation related to the appropriate data model element. At the same time users and developpers can check in the data model, the meaning of a Utype, data type, etc.

5 Utypes Syntax

The building-up of a Utype string has been discussed extensively within and between the Data Model WG, the DAL WG and Theory IG and at various IVOA Interoperability meetings. Here is a proposed syntax we have agreed on for a simple valued element. See also the Utype section in the Simulation data model IVOA standard [5].

The Theory interest group has come up with a minimal, necessary set of rules to produce a string that uniquely represents any of the fundamental syntactic elements in the model. These rules are the following:

- Property names are unique in a Class. Note there are three types of properties:
  - An *Attribute* is a property the datatype of which is a value type (NOT an object type/class), though it need not be primitive but may be structured (i.e. have attributes of its own).
  - A *Collection* is a named, 1-to-many composition relation of a parent to a child class.
  - A *Reference* is a named, many-to-one shared association to another class.
- Class names are unique in a Package.
- Package names are unique in either an enclosing parent package, or in the set of models adopted in the IVOA.

So a name like (in a pseudo regexp notation)

```
<model-name>:[<package-name>/]*<class-name>.<attribute-name>[.<attribute-name>]*
```

is a unique pointer to an attribute in a data model. Note that most of data models in the IVOA include all their necessary classes in only one package, then the package name can simply be ommitted. Similarly

```
<model-name>:[<package-name>/]*<class-name>.<reference-name>
<model-name>:[<package-name>/]*<class-name>.<collection-name>
```

are unique pointers to the reference and collection properties of a class. *When classes are embedded, there may be attributes before the reference.How do we handle this?*

The rule allows for an arbitrary nesting of packages, which is necessary to ensure a unique encoding. Since attributes can be structured, we allow for chaining these until the final primitive attribute is reached, i.e one which carries a single value.

The reference name (resp. collection-name) is an explicit name for a link, or pointer to a target class, as shown in fig. 2 where *SimDM:/resource/experiment/Experiment.protocol*, is a reference to the protocol used to realize this particular Experiment.
to improve ...

In the Utype string construction, references and collections are NOT followed further. Only the pointing mechanism is expressed in the Utype. The referenced (target) class will be encoded normally and pointers will be implemented to it.

Each serialisation mode can support this:

- XML will use the ID/IDREF mechanism to set a link from the class to the referenced one provided the two connected classes are defined in the same document.
- VOTable applied the same mechanism.

Reference could be implemented in the (Utype,value) pair list, but XML and VOTable are much more convenient for that. Therefore the Utype list serialisation should be reserved for small sets of metadata consisting in single value attributes, as used in the various VO protocols.

The path can be stopped at any level in the UML acyclic graph, which allow a Utype to refer to a class, and not only to a lower level attribute,

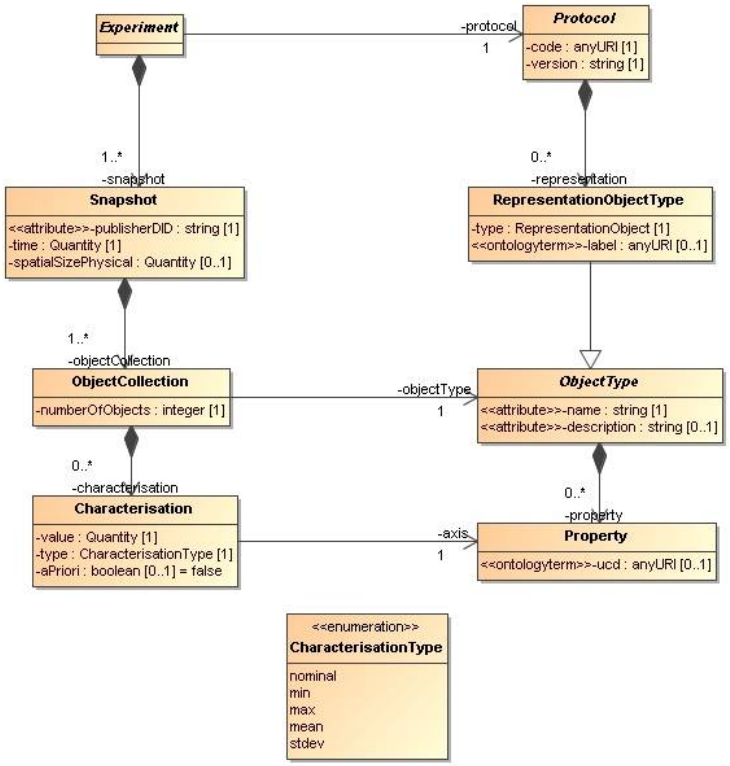

Figure 2: *UML diagram excerpt of the Simulation data model. This illustrates the reference mechanism between classes. Here* ObjectType *is the name of a class, that can be accessed from the* ObjectCollection *via a link or pointer called* `objectType`*. This means that the class ObjectCollection gathers objects whose types are described in the ObjectType class, allowing to consider any new types of objects.*

with a single value. These constructs are interesting in hierarchical VOTable serialisation, where by using a GROUP structure we can fully encode nested objects. See the VOTable Example in Appendix B and coordinates serialisation in [3].

6 Data model re-use

6.1 From Classes of existing IVOA DM

In the VO there are common structures that are needed everywhere, like IVOA identifiers, or coordinates. Coordinates are defined in a separate model: STC, http://www.ivoa.net/Documents/latest/STC.html and identifiers are standardised in http://www.ivoa.net/Documents/latest/IDs.html. Data identification and curation metadata are also reused from VOResource, Spectral DM, etc., so we enconter situations where a class or a set of classes, defined in one DM are re-used in another one.

In object oriented programs, classes of these packages are simply linked using libraries, and can then be used as types (primitive classes) for other models. For serialisation, we need an explicit mechanism to mention that attributes in a class re-use basic structures from another DM, as STC classes for instance.

In order to clearly trace back the definition of classes re-used from a data model we opted for the re-definition of these inside the new model, like a copy of existing classes. This first allows to properly define the re-used elements in one version of the model especially if they underwent several evolution steps through various versions of the models. This also supports class derivations or associations explicitely.

Therefore the UML class diagram explicitely show re-used classes from data models, either as advanced datatypes for attributes or as referenced class. For instance in the ObsCore DM [6], DataID and Curation classes are re-used from SpectralDM as classes and pointed to from the Observation class. The AstroCoordArea class from STC is re-used as a data type for the 'area' attribute in the 'Support' class. The XML schema definition for a DM1 including and/or re-using elements of DM2 will import the necessary XML schema for DM2 and refer to the re-used parts via the XML name space mechanism. See Obscore XML schema definition at http://www.ivoa.net/xml/ObsCore .

The Utype list list definition on the contrary, does not need to materialise the border between both models. It is built up from the natural concatena-

tion of the utype string in the top level DM (Characterisation DM in fig. 3) below with the utype of the re-used element from the second data model (STC in the example).

```
<SpatialAxis>                          Corresponding UTYPE
<axisName>spatial</axisName>
<ucd>pos</ucd>
<unit>deg</unit>
<coordsystem id="TT-ICRS-TOPO"         <---->cha:SpatialAxis.Coordsystem
xlink:type="simple xlink:href="ivo://STClib/CoordSys#TT-ICRS-TOPO"/>
<coverage>
<location>                             <---->cha:SpatialAxis.coverage.location
    <coord coord_system_id="TT-ICRS-TOPO">
        <stc:Position2D>
          <stc:Name1>RA</stc:Name1>
          <stc:Name2>Dec</stc:Name2>
          <stc:Value2>
          <stc:C1>132.4210</stc:C1>    <---->cha:SpatialAxis.coverage.location.Position2D.Value2D.C1
        <stc:C2>12.1232</stc:C2>
          </stc:Value2>
        </stc:Position2D>
    </coord>
</location>    ...
</coverage>
</SpatialAxis>
```

Figure 3: *Utypes are simply built by recording the path while browsing the XML tree down to the leaves.*

The concatenation is supposed to happen only once which means the rightest part of the Utype string belonging to DM2 and showing the finest levels of nesting are some kind of VO types described consistently and self sufficiently in some data model. This makes the assumption that VO models are properly organised in nested packages and are cooperative enough to cover the whole field of astronomical metadata with a minimum of overlap.

If we consider for instance a specific version of the Characterisation data model whose classes integrates coordinates and regions from the STC v1.33 data model, we get a simple notation by just browsing down the attributes chain as shown in the syntax section.

```
char:SpatialAxis.Coverage.location.Position2D.Value2D.C1
```

Such a notation does not show the limit between the two models but is consistent with the XML schema import mecanism. Interpreting the data-model name 'char:' by parsing the Utype string and resolving the corresponding data model link will point to the specific version of the CharDM with the specific STC v1.33 data model version. This provides a tight binding between the two data model versions. From the Utype string, the nested classes involved in both models can be rebuilt.

In the special case of VOTable serialisation, widely used in VO tools and astronomical applications, specific rules have been defined to reference STC coordinates using the GROUP construct. See [3] for details on the STC Utypes in VOTable GROUP structures. STC components are gathered in groups with their Utype stating explicitely the 'stc:' prefix. This allows STC-aware tools to deal with coordinates independantly of the outer structure and facilitates library support in code generation.

Generalising the usage of GROUP is a way to identify re-used data model components.

6.2 Data model extension via customised Utypes

The Utype definition should support data model extensibility. We identified several cases where some metadata handled in an archive or an application are partly covered by an IVOA data model and some other metadata are not covered at all by any IVOA model. Then the data model could be extended for a specific purpose, on a case to case basis. The data provider or application developer should find a possibility to reuse the data model parts and add new features. These customized features should be added in a customized version of the model , by inheritance as much as possible , and not overlap with existing properties. The developer could define new Utypes for the customized data model fields. New utypes should be built with the same procedure as for recommended data model (syntax, object inheritance, symbols). They can be distinguished from the original model by a customized data model name like in the following example. Suppose we want to reuse concepts of the SED DM , concerning the flux value: the DM has

```
sed:Data.FluxAxis.value
```

but for some specific data centers like NED, a slightly different quantity is computed specifically and provided. The NED service can then extend the SED service and produce a new data element with utype for instance like: sedNED : Data.FluxAxis.publishedValue

```
sedNED:Data.FluxAxis.publishedValue
```

This allows to identify immediately that a metadata tagged like this belongs to another model and hook to its particular definition. This also needs to produce some documentation about the value type, units, ucd, etc., attached to this new data model element.

These two examples of data model reusability suppose that data models in the IVOA are properly documented and can offer machine-readable infor-

mation for implementers. It is desirable for a (utype,value) pair to be able to recognize the utype in a data model, to point to the data model element definition and check or interpret units, data type, etc., in order to use the value in a compliant fashion.

We can then derive that a datamodel should have :

1. a utype list published together with the IVOA standards document
2. an on-line documentation where the description of each data model item is specified as required in section 4.

When an application uses a utype-based serialisation instance, it should :

- Check the correctness of the Utype string , by searching it in the simple Utype text list. String matching is more effective in small case to avoid mis-spelling . However the CamelCase adopted in the early definition of Utypes (Spectral DM, CharacterisationDM) make them easier to read for the user or implementer.
- Use the Utype as an anchor in an on-line HTML document to provide help functionalities and point to the data model element description.

This is implemented for instance in the Simulation data model documentation available at
http://volute.googlecode.com/svn/trunk/projects/theory/snapdm/specification/html/SimDM.html and follows a suggestion by N.Gray [4] which offers a way to go for integrating more automatic metadata identification.

7 Generating Utypes from UML data models via their XML representation

The syntax rules proposed in Section 5 above can be implemented from an XML schema representing the data model, using the XPATH mechanism [8] to build up a path from the root of the schema down to the finer grain elements corresponding to attributes' class in the model. XPATH is not directly used in Utype generation , but its properties are indirectly applied in the approach described here.
Suppose now that we have an XML schema fully mapping the UML model content, with all classes represented as elements in the model, nested elements for aggregation, references and basic types.
For the sake of clarity, we do avoid substitution groups and choice patterns and on the contrary prefer the XML extension mechanism. Such a rule

helps to guarantee that for one XML element at any level, its name can be mapped to only one sub-structure and therefore allow for direct class encoding. Nested classes will be organized as XML trees, then browsing the tree down to leaves elements and concatenating the names at each level provides a path which is similar to the Utypes construction mentioned in the previous section(cf 5).

In order to achieve a proper mapping from UML to XML serialisation, and derive object code or Utype list from the generated XML, some requirements on the style of UML design as well as the XML schema construction should be met.

- UML : For any association , each class connected should have a role name in order to clearly identify references. Template classes provide a same name for different typed structures and are difficult to translate in XML; hence they should be avoided.
- XML Classes, should be converted as XML elements and class attributes as included sub-elements. The XML attributes are more or less providing context for the XML translation and are not used to describe the data model structures(only valid for charac. simdb has a diff. strategy).

Most of the UML modeling commercial tools like RationalRose, MagicDraw, Modelio , etc... have an internal XML representation of a UML model encoded in a proprietary XMI format. When simplifying this representation, one can apply XSLT transformation rules to directly generate output products like :

- a set of hyperlinked webpages for the datamodel documentation
- an XML schema
- an example of XML document instance
- a Utype list with documentation

Such an approach has been implemented with success by G. Lemson and L. Bourges in the Theory interest group. see http://volute... and described as the VO-URP project.

It provides a pipe-line of transformations starting from the '.xmi' file and producing the various artefacts: Utype list, documentation, XML schema.

UML allows various designs for a specific project and fully integrates the properties of graphs, with association links between classes while on the contrary XML emphasizes the hierarchy of elements. Therefore the translation is not straightforward. Some modeling rules should be imposed in UML design in order to simplify translation and produce robust XML schema and Utypes list.

However, although they rely on XML language, each commercial tools defines its own profile for the definition of its XMI formats. Therefore the tuning of the VO-URP tools for a different framework needs a carefull adjustment and is not straightforward. Portability from MagicDraw to Modelio was particularly painful.

The documentation for a Utype is defined when the data model is built up and stored in the XMI representation of a UML Model. Most case tools provide a documentation generator that produces an HTML hyperlinked set of pages for the datamodel documentation including UML diagrams. These may contain just a set of few lines or a full illustrated text if necessary. N. Gray has proposed an URI generation function for each Utype in a DM, that could be used to point to the corresponding anchors of the on-line documentation of a data model.

8 How are Utypes used?

8.1 Publishing data to the VO

Up to now, data models like SpectrumDM, CharacterisationDM, SimDB data model help to define, represent and manipulate metadata. They provide UML diagrams, XML serialisations and Utypes lists for the model classes.

Within the DAL WG, protocols such as SSA also make use of Utypes. The SSA protocol version 1.04 has its own Utype serialisation attached in Appendix D: 'SSA Data Model Summary' of the standard document [1] .

Data Providers can use Utypes to label the metadata attached to their data collections. The process will be the following:

- select a data model which covers the usage domain of these data
- map proprietary metadata (FITS, Archive, Etc..) to the Utypes of the selected model
- generate metadata as serialised documents (VOTable, Utypelists, others?)

Different scenarios can be explored : to be developed: To publish data with the CharacterisationDM-v1.11 , one can use the CAMEA VO Tool (http://eurovotech.org/twiki/bin/view/VOTech/CharacEditorTool) to check the Utype assignation, and verify if the Utype serialisation is compliant to this model.

At the data collection level, tools have been developed to help for keyword mapping from FITS keywords to Utypes list: Here is a list of the first tools developed for that:

- FITS to DAL interface or data model Utypes:

- MEX (ESO) DAL interface link...
- DM-Mapper (ESA) DAL interface link...
- Interactive mapping tool (CDS) (prototype) link...

These tools take a data model description from the IVOA and help the data provider to interactively build a map table from FITS keywords to Utypes. The mechanism is extensible to any kind of data model supporting the documenation and utype list required. It would help data providers to map their metadata to a standardized VO Utype description and let VO tools operate access to them.

8.2 Naming and identifying metadata in VO protocols

Broadly used protocols like SSA or ObsTAP include Utype mapping. The SSA query response consists of a number of fields, identified by Utypes, as defined in the data model summary part of these standards. See standard documents at http://www.ivoa.net/Documents/latest/SSA.html , Appendix D and Appendix B in http://www.ivoa.net/Documents/ObsCore/index.html,
.

Similarly the SLAP protocol defines its own set of Utypes in the Appendix D of the Simple Spectral Line Access Protocol V0.9 standard document(http://www.ivoa.net/Internal/IVOA/SpectralLinesListDocs/WD-SLAP-0.9-20090518.pdf
). More widely the VAMDC project also provides the same kind of mechanism to refer to its data model items. Check at
http://dictionary.vamdc.eu/restrictables/ .

8.3 Querying data bases

Queries in ADQL or SQL use column names to ask for information. For a data base to be compliant with a data model, only the mapping between existing columns and Utypes must be defined. Thi can be done internally inside the data base tables , but also via a mapping table as used in the `TAP_SCHEMA`table definitions. A simple scenario could be the following:

1. the client application asks a server for its list of supported metadata and Utypes (mapping table)
2. the server exposes the mapping
3. The user selects the metadata he/she requires by browsing the Utypes and the accessible documentation.
4. the client translates each Utype in the query into a column name and submits the query

5. the server parses and resolves the query and sends back the results columns
6. the client translates each column name in Utypes when possible and displays the results.

Such a scenario is interesting as if offers a general vocabulary to the user, whatever the data base content and needs few steps of re-engineering.

9 Conclusion

Utypes are useful to convey the role, the structure and the normalized name for each piece of metadata involved in an IVOA service or protocol. It supports the various use-cases The syntax is derived from the UML design of the related data model. It is an important factor in interoperability. A compromise between long descriptive strings and usability has been found in developing simple mapping mechanism at the client side.

References

[1] Tody D. Et al. Simple spectral access protocol. *http://www.ivoa.net/Documents/latest/SSA*, 2007.

[2] D. Crockford. The application/json media type for javascript object notation (json). *http://tools.ietf.org/html/rfc4627*, 2007.

[3] M. Demleitner, F. Ochsenbein, J. McDowell, and Rots A. Referencing stc in VOTable. *http://http://www.ivoa.net/Documents/Notes/VOTableSTC/*, 2010.

[4] Norman Gray. Utype proposal. *http://nxg.me.uk/note/2009/utype-proposals/*, 2009.

[5] G. Lemson, H. Wozniak, and al. Simulation data model version 1.0. *http://http://www.ivoa.net/Documents/SimDM/20120302/*, 2012.

[6] M. Louys, F. Bonnarel, D. Schade, P. Dowler, A. Micol, D. Durand, D. Tody, L. Michel, J. Salgado, I. Chilingarian, B. Rino, J. de Dios Santander, and P. Skoda. IVOA Recommendation: Observation Data Model Core Components and its Implementation in the Table Access Protocol Version 1.0. *ArXiv e-prints*, November 2011.

[7] Francois Ochsenbein and Roy Williams. VOTable format definition version 1.2. *http://www.ivoa.net/Documents/VOTable/20091130/*, 2009.

[8] W3C. Xml path language (xpath) 2.0. http://www.w3.org/TR/xpath, 2007.

A Appendix A: Utype serialisation example

*i*nclude Omar examples for serialisations

B Appendix B: Serialisation examples

TBC Try to have a data set described with 3 serialisation format:
"utype+value" liste, VOtable, XML instance

Example: CADC ObsTAP service query response to a the query "SELECT TOP 10 * FROM ivoa.ObsCore " The query response to this query is returned in a VOTable, with all fields tagged with Obscore Utypes. See an extract of this VOtable here :
http://www.ivoa.net/internal/IVOA/Utypes/ObsTAPQueryExample.xml

C Appendix C: Example of a data model Registry entry

Here is a possible example of data model entry, in XML , inspired from the StandardregExt service record. This could be used as introductory part for any instance document containing a serialisation of the related data model. This is a complete reference to IVOA documentation for a data model so usable in any serialisation format.

An example of a registry record defined for the Photometry data model is available at:
http://www.ivoa.net/internal/IVOA/Utypes/PhotDMRegistryRecord.xml

D Appendix D: Updates of the document

- version 0.3 to 0.4
  - introduce canonical and alternative notations
  - update fig.1 and fig.2