Astronomical Data Analysis Software and Systems XIII
ASP Conference Series, Vol. 314, 2004
F. Ochsenbein, M. Allen, and D. Egret, eds.

Data Models in the VO: How Do They Make Code Better?

Norman Gray, David L. Giaretta, David S. Berry, Malcolm J. Currie, Peter W. Draper, Mark B. Taylor[1]

Starlink Project, Rutherford Appleton Laboratory, Chilton, Didcot, OX11 0QX, UK

Abstract. Data Models exist in people's heads. Data modelling consists of making these explicit on paper, so that (a) we can discover if there is more than one important model, and (b) we can develop using the model which has the best impedance match with the targeted community.

We contend that there is in fact more than one model relevant to the Virtual Observatory (VO), and that while the VOTable model is a valuable fit to the archivists' model of data, it may be a poor match for many users or (much the same thing) for the software written to service the sort of end-user astronomical applications which the VO targets.

We will also review some of the various modelling languages available.

1. Language, models and usability

In linguistics, the well-known Sapir-Whorf hypothesis claims that the way we conceive of the world depends on the language we use to describe it. This means that the language we use affects what we can think; and conversely, if we wish to have and manipulate a thought, we must find some language to express it.

All this talk of languages matters to us, since when we create software systems, we are generally creating a 'language' – in a user-interface, an API or a protocol – which users must employ to interact with the underlying system, be it an application, a library, or a remote service. The user, the language and the system each have a *model* associated with them (see Figure 1), and if there is a good three-way match between the models, then the user's interactions with the system will be straightforward and generally error-free. If there is a mismatch, they will not. This is not just a matter of user-interface design; if the 'user' is a programmer using an API or protocol, then this three-way match will help them write correct code faster, and help produce an application which will be usable by its eventual mouse-wielding audience.

In some cases this match is reasonably obvious (think of the web and either HTML or HTTP); in other cases more work is involved (Unix shell language is tightly bound to the underlying system, but it takes effort for the user to

[1]Alternate affiliations: NG, University of Glasgow; DSB, University of Central Lancashire; PWD, University of Durham; MBT, University of Bristol

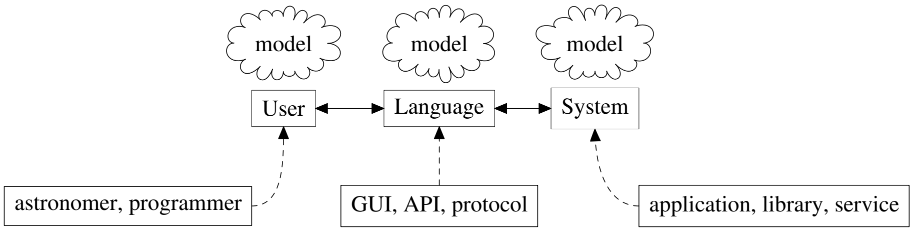

Figure 1. Systems, users, and the languages which mediate between them, all have potentially separate implicit models.

acquire the corresponding model); and in other situations (notoriously video recorder interfaces) the complete dislocation between the three models makes the interface language almost unusable.

Ideally, then, there is a single model, which our user thinks with (possibly with the help of documentation), which the language expresses, and which the underlying system implements; it is the rendezvous which helps the system as a whole hang together. If the model is made explicit during the development process, then it can itself be examined, criticised, and checked for consistency with itself and with the external system it is supposed to model, whether that is an archive or a 'quantity'. The Sapir-Whorf hypothesis suggests that it is only once the model is itself part of the language of development, that we can think with it and talk about it. Thus this is a software quality and usability issue; it is an abstraction with the concrete goal of freeing software design from the details of any particular implementation.

It is important to note that the interface language's syntax is not a model. Instead, that syntax should be chosen so as to faithfully reflect the model which the language hopefully shares with user and system. However, we can only discuss the faithfulness of the syntax once we have an explicit model.

So how do we make the model explicit? There are several obvious answers to this, of which the most currently fashionable will be XSchema, UML, and 'a Java class library'. A potential problem with each of these is that they each come with a good deal of baggage. This is another aspect of the Sapir-Whorf hypothesis: we are driven to see the world in terms of the structure of the language we use to describe it, irrespective of whether these features are present in, or adequately describe, the system being modelled. That means that if we use XSchemas to model the world, we will discover that the world is hierarchical with attributes, and if we use an OO language, the world turns out to have methods. For example, while it is certainly possible in XML to model circular reference or multiple containment (an RA IsA quantity *and* IsA position), the resulting language would likely be confusing and hard to use correctly.

There are several strategies to deal with this. The first is to decide that syntax is more important than anything else and let that lead the process (this was arguably the case for UCD1). Another is to confront the problem, acknowledge that our choice of language is not neutral, and make sure that choice is a good one. A third is to choose a more primitive modelling language, which will push fewer things into the model. We will return to this question in Section 3.

2. Different models for the same data – the case of VOTable

Our second point was first made by the Emperor Charles V: "I speak Spanish to God, Italian to women, French to men and German to my horse". If there is more than one type of user, and thus more than one user model, we may need more than one language to achieve the required three-way match.

Several of the systems in the current VO use VOTable syntax (Williams et al. 2002). Despite this, we claim that they do not all use the underlying VOTable data model. VOTable is excellent as a way of archiving catalogue metadata, and encoding *all* the available information about a catalogue or image. This comes about because VOTable is expressive, recursive and flexible, and these are advantages for many users, with the result that there is a good match between the user, system and (VOTable) language. There is another category of users, however, who do not need or want this sophistication, and who simply wish to extract a more modest amount of information (such as image and variance data) with as little knowledge as possible; this may be because they are busy and cannot afford the time to read full documentation, or because they are writing generic applications, and so cannot afford the luxury of reading documentation and building in to their application knowledge of the various ways that a set of data providers have exploited VOTable's flexibility. The Simple Image Access (SIA) protocol implicitly uses this simple model at both the user and system ends, even though it uses in its responses the VOTable syntax. This dissonance between models is a potential usability problem, and is addressable by using an alternative model such as that of HDX (Giaretta et al. 2003).

The UCD system (Derriere et al. 2004) implies a third distinct data model, corresponding to a distinct constituency which is interested in metadata rather than pixels, and whose focus is on registries rather than image viewers.

This (Kuhnian) incommensurability is not a defect which can be evaded by a yet more comprehensive Grand Unified Data Model; instead, it is a fixed feature of the problem that the VO is addressing. Although they are less severe, there are also similar dislocations between the metadata models which different VO participants prefer. These gaps between models can be bridged by software that understands more than one model (such as an archive which can service two communities), by a consensus model, if one can be produced in a useful timescale, or by accepting the existence of multiple models, and declaring what mappings exist between them in such a way that software can mechanically extract the semantics of a given set of metadata with at least as much fidelity as it would from a consensus model. It is this last which seems the most flexible and scalable of the options.

To illustrate what is gained by being able to discuss the data model explicitly, consider the following pair of RDF statements, which is how we might express the fact that a column with ID `raerr`, say, is the error in an RA:

```
_x rdf:type :pos.eq.ra .
_x :stat.err #raerr .
```

This is RDF 'notation3', and expresses in detail that there exists a thing `_x` that is of type `pos.eq.ra`, and has a property `stat.err` which is in the URL `#raerr`; however the details are much less important than the observation that such a neutral notation exists, and that this notation is distinct from the language we would use to communicate this in practice. This prompts a number of valuable questions: `pos.eq.ra` appears to be a type whereas `stat.err` is a property (in RDF terms): did we want this? Is this what we want a UCD like `stat.err;pos.eq.ra` to mean? If not, what? For a given proposed UCD syntax, what would such a set of statements look like? Can we construct similar sets of statements which our proposed syntax could not express? Such questions are extremely difficult to ask, let alone answer, without a notation which is distinct from the syntax under discussion.

In summary, the VO contains multiple user groups and models, and its challenges therefore cannot be met by an approach focused on a single central model. Even when groups have largely compatible models (such as the producers and consumers of complicated archive data), the mappings between them, and consensus models, must be discussed using a notation *which is distinct from the proposed syntax*, if it to be possible to discuss that syntax with clarity.

3. Modelling techniques

RDF, illustrated above, is one of several possible modelling languages.

XSchema is the World-Wide Web Consortium's (W3C) standard schema language which, as well as validation, is usable is a modelling language. It is both verbose and rather complicated, and its main benefit over DTDs is its elaborate type system, which looks familiar to those with a database schema background, as well as being the type system for a variety of other W3C standards.

The Unified Modelling Language (UML) is intended for designing object-oriented systems, and modelling their environments. Object-oriented notions of subclassing and object manipulation are very natural within UML, in the way that containment, for example, is natural in XML schemas. The Object Management Group (`www.omg.org`) curates the UML standard as part of the larger Model Driven Architecture (MDA) as a means of specifying platform-independent application descriptions: XMI (XML Metadata Interchange) is an XML-based modelling language intended to help generate and exchange consistent XSchemas, UML, and code.

Resource Description Format (RDF) is a W3C low-level modelling notation, which is usable by the inferencing engines which should drive the Semantic Web; it is the modelling aspect, rather than this inferencing one, we have emphasised above. Other W3C standards such as RDFSchema and OWL supplement the base RDF syntax to the point where it is useful for describing realistic ontologies.

References

Derriere, S. et al. 2004, this volume, 315

Giaretta, D., et al. 2003, in ASP Conf. Ser., Vol. 295, ADASS XII, ed. H. E. Payne, R. I. Jedrzejewski, & R. N. Hook (San Francisco: ASP), 221

Williams, R., et al. 2002, VOTable: A Proposed XML Format for Astronomical Tables, version 1.0, `http://cdsweb.u-strasbg.fr/doc/VOTable/`