VO-DML from a VODSL Perspective

Paul Harrison - Jodrell Bank

Asterics Tech Forum 2, Edinburgh 2016

Contents

- Brief overview of VO-DML
- What is VODSL?
  - Introduced at Madrid IVOA Interop 2014
- Remarks about VO-DML
- Remarks about current data models
- VODSL Eclipse demonstration (if time allows - otherwise see hackathon)

VO-DML

- The problems with pre VO-DML data models.
  - Need to attach a label data item - UType was invented - however exact relation to existing data models was not rigorously defined.
  - Data model reuse difficult
    - Dtandards often only considered their own domain - led to repetition and overlap
    - Not machine readable
      - Often only representation available was UML diagram and/or list of UTypes in standard
      - Sometimes XML schema available, but that often not an exact representation of the model because the quirks of the XML schema language
- VO-DML solves these problems by defining a subset of UML with an XML serialisation
  - Tools to generate UTypes and XML schema mechanically from VO-DML.

Motivation for VODSL

- Creation of VO-DML seems to have high activation energy (only a few people are doing it)
- UML route
  - Different tools have poor interoperability and can be expensive
  - UML very general - profiles can help
- Direct VO-DML XML editing route
  - Does not *need* fancy tool
  - Much better constrained - can only express the concepts that are desired *but* still not very human friendly for writing by hand

The University of Manchester
Jodrell Bank
Observatory

VODSL

- concise cf XML
- sensible defaults
- fully expressive
  - yet more constrained than XMI/UML

direct re-use

extends

```
model example (0.1) "an example model" author "Paul Harrison"
include "IVOA.vodsl"

primitive angle "a new primitive"

package apackage "example package"
{
        dtype fq -> ivoa:quantity.AtomicValue {
            value : ivoa:real "a new float";
        }
        abstract otype base {
            bv : ivoa:boolean  "Description";
            ^author: ivoa:string @? "author"; /* note use of ^ to be able to
                                                 re-use reserved word.*/
        }
}
otype derived -> base{
    sv : ivoa:quantity.RealQuantity "";
}
otype another {
    f1 : apackage.fq @[6] "an array";
    f2 : nestedpackage.another as composition "";
}
package nestedpackage
{
    otype another "" {//same name different package
        cc : derived as composition "" ;
        rf references apackage.another "a reference";
    }
}
```

comment

multiplicities

mandatory description

VODSL implementation

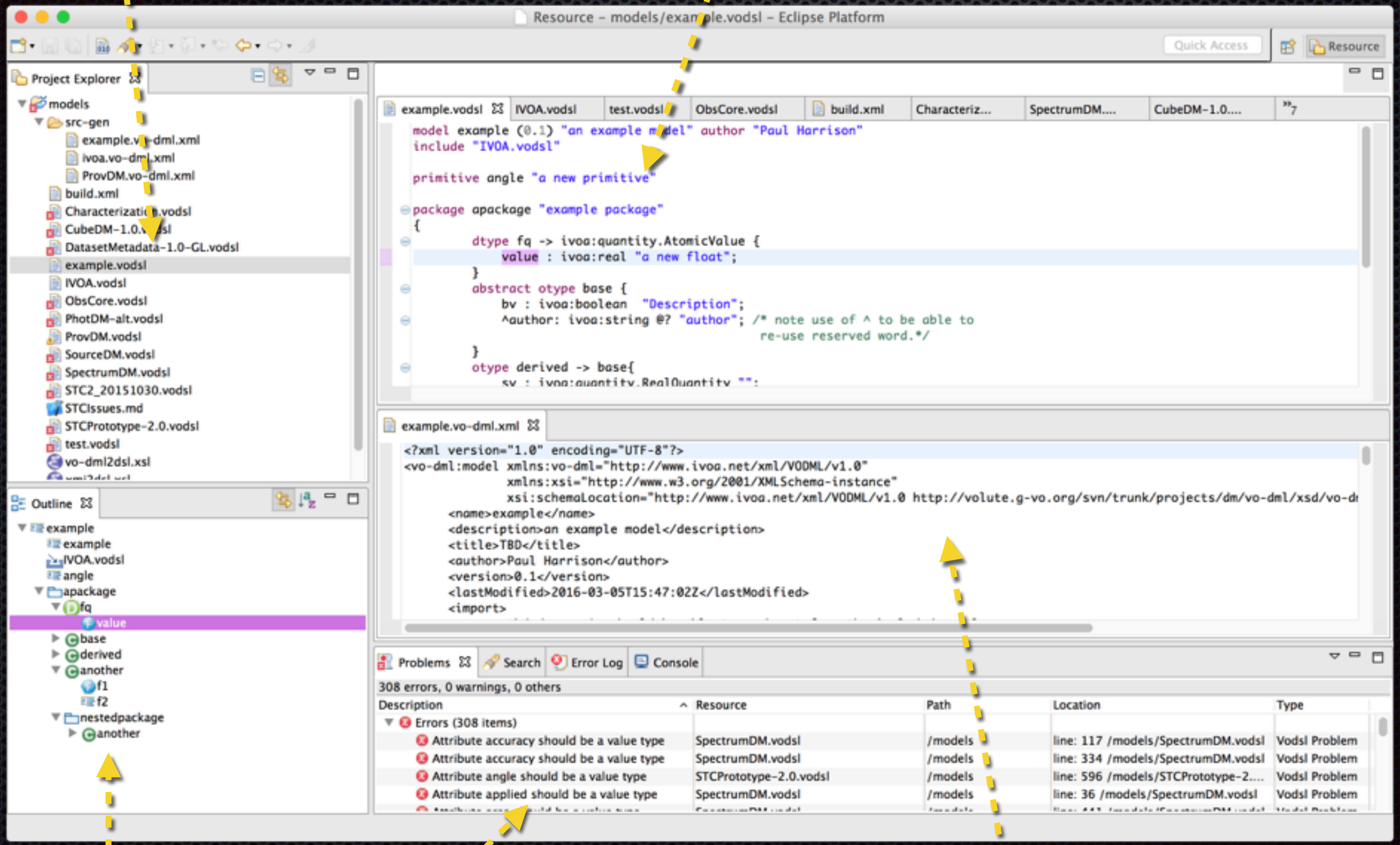
- Using Eclipse Xtext - a language development package
  - easy to create a domain specific language (DSL) with full Eclipse editor functionality
    - syntax highlighting/formatting
    - auto-completion
    - validation
    - quick fixes
  - Just write the grammar and the much of the above comes for free! (it can be further customised if the default behaviour is not quite what is desired).
  - compiles to the VO-DML XML form for compatibility with existing infrastructure
- Can be regarded as a necessary reference implementation of VO-DML for standardization!

The University of Manchester
Jodrell Bank Observatory

Typical Eclipse Workspace

file browser

VODSL editor window



outline structure

validation errors

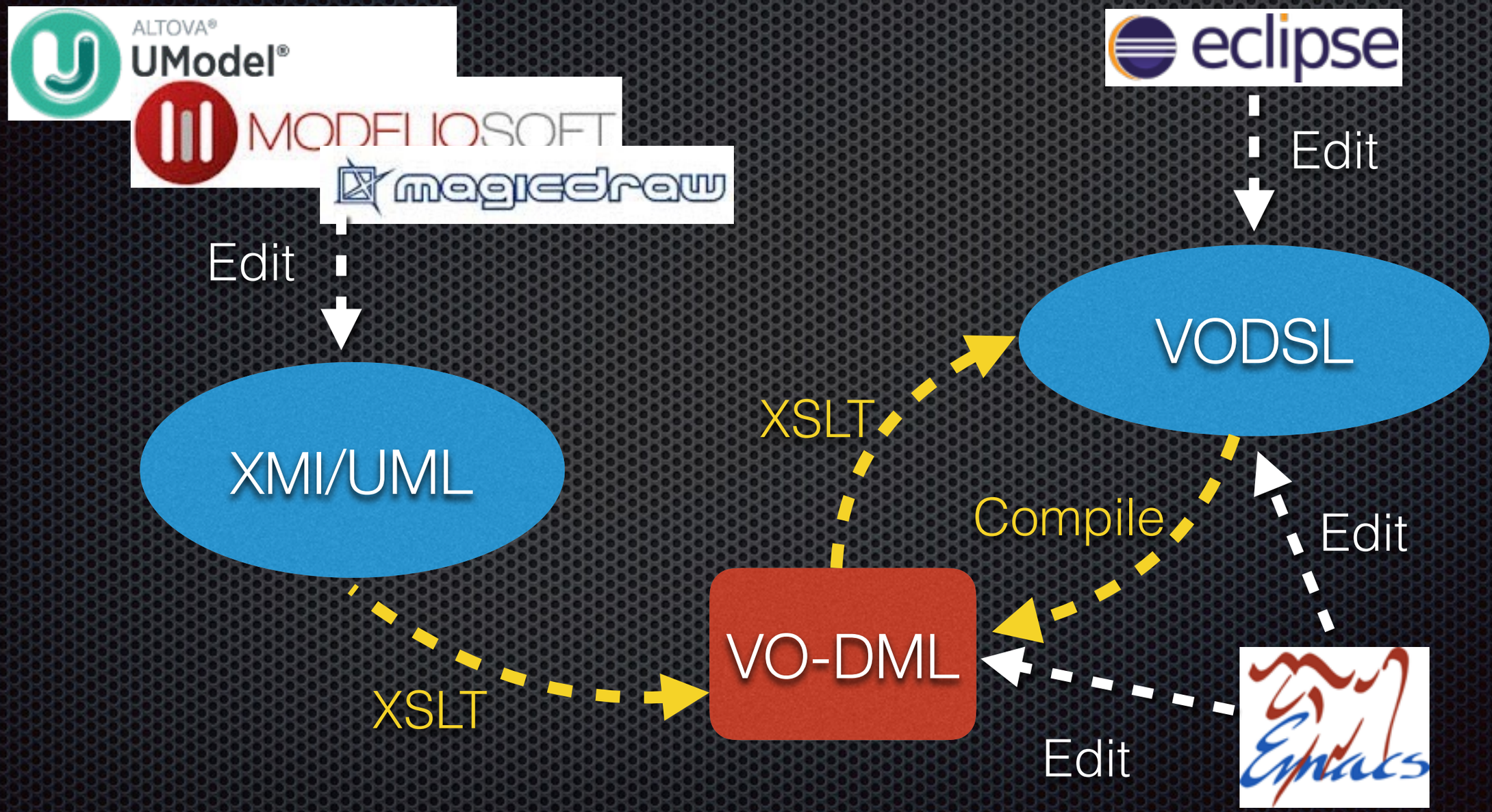
generated VO-DML

VODSL updates

- Follows changes in VO-DML specification
- More natural scoping - not always necessary to fully qualify names - also use of ':' as namespace separator for model level
- Validation - most of the schematron rules now implemented and work as you type!
  - missing "unique composition" and "subsetting" rules
- Some grammar and syntax changes
- Round-tripping to/from VO-DML generally more robust.
- Available on GitHub https://github.com/pahjbo/vodsl along with models translated from volute
  - Can supply as precompiled Eclipse plug-in
  - Could be packaged as a stand-alone compiler VODSL->VO-DML

Ecosystem

- VODSL complements rather than competes with VO-DML



VODSL route vs UML route

VODSL		UML
Easier to perform global refactoring	vs	Easier to visualise the whole model
Instant validation	vs	Full validation only after XSLT transformation of XMI
Easier to merge contributions from two authors textually	vs	Rely on UML tool to have model merging facility

VO-DML remarks

- Standard document feels aimed more at VO-DML implementers than data model creators.
  - Document structured around XML representation
- There are still some issues with the language itself
  - Subsets seem a little loosely defined
  - Restrictions on attribute multiplicities - though recent mailing list update seems to have relaxed this
  - Aggregation pattern

Utility of Data Models?

- What use are they?
  - Understanding what an individual data item is ✔
    - Looking at the relations between data items
      - enables you to make meaningful queries (VOQL?)
    - Provide storage schema
- Automated software generation from models

Data Model Design

- Even with the aid of sophisticated tools, the design of the data models takes great skill
  - Desirable to cause as little disruption as possible to existing models
    - There are some well known UTypes in widespread use.
- Optimal design of model affected by the purpose for which it is used
  - Not an implementation model for a particular computer language
  - But not a just a domain model - the data pigeon holes for data are the most important concern.
    - But deep object trees are not necessarily helpful - especially when it comes to code generation.

2nd Generation Data Models

- There has been an effort to regularize models in VO-DML and the progress is being stored in the volute repository
  - Cube
  - STC2
  - Provenance
  - DatasetMetadata
- I have converted latest models from volute and put in VODSL GitHub project.

Data Model remarks

- Reuse still not fully established via import - sometimes models have ‘copy and pasted’
- Still many errors in the raw translations
- Problems with original models?
- Problem with XSLT->VODSL?
- STC2 might be the most rigorously correct VO-DML model currently existing!

Conclusions

- I believe that the Eclipse VODSL tooling allows another route to creating VO-DML models that is at least as good as the existing methods.
  - Provides an additional way to view and check whether models are “well designed”
- There should be a number of ways by which data models should be judged
  - Generated UType structure
  - Generated code
  - Generated relational schema
- It would be good if there were more guidelines for data model creators to provide for a more uniform style as well as more endorsed ‘patterns’