Issues in Mapping Metamodels in the Ontology Development Metamodel Using QVT[1]

Robert M. Colomb[2], Anna Gerber[3], Michael Lawley[3]

colomb@itee.uq.edu.au, agerber@dstc.edu.au, lawley@dstc.edu.au

Introduction

Although the request for proposal for the Ontology Development Metamodel called for a metamodel and UML profile supporting OWL, the working party has decided that there are a number of modeling languages used for ontology development so that several languages will be supported in the ODM [odm], namely RDF/RDFS/OWL, Simple Common Logic (SCL), Topic Maps and the Entity-Relationship model, as well as UML.

There are several ways to integrate several metamodels in one structure:

- Designate one metamodel as primary and represent the others by subclassing
- Designate one metamodel as primary and translate the others into it
- Keep the metamodels separate and map them into one another using QVT [qvt].

The first two ways were thought undesirable because they would inevitably distort the metamodels not designated primary and would unduly complicate the primary metamodel. Since QVT can retain the association between the source and target model elements in a mapping, the mapping approach allows each metamodel to retain its integrity. To avoid an n^2 mapping specification, the ODM working party decided to define a star configuration of bidirectional mappings with a single metamodel as a core. The core should not be one of the designated metamodels since they are autonomous, but since OWL was central to the RFP, the core was chosen as an abstract syntax for a published version of Description Logic.

Translation between metamodels has the fundamental problem that there may not be a single and separate model element in the target corresponding to each model element in the source (indeed, if the metamodels are not simply syntactic variations, this would be the normal situation). We will call this situation *structure loss*. This is why QVT maintains traceability between source and target elements. In the worst case one can always see where a given model element instance came from to help interpret it. But if we are going to work in the target metamodel we want to refer to the source as little as possible. The purpose of this paper is to explore some of the reasons for structure loss, looking at what difference the loss makes and therefore how serious the problem is in particular cases.

Our work has not progressed sufficiently that we have any claim to a comprehensive theory of structure loss. We will look at three particular situations to see some of the issues:

- UML and ER have two elements mapping into a single element *property* in DL or OWL
- SCL is a very much stronger theory than any of the others
- OWL Full allows classes as individuals, a second-order structure, and none of the others do

[1] The work reported in this paper has been funded in part by the Co-operative Centre for Enterprise Distributed Systems Technology (DSTC) through the Australian Federal Government's CRC Programme (Department of Education, Science and Training).
[2] School of Information Technology and Electrical Engineering, The University of Queensland, QLD 4072 Australia
[3] DSTC Pty Ltd

Using QVT map between the models

For the purposes of this paper, we are using DSTC's QVT transformation language (described in DSTC, IBM and CBOP's second revised submission to MOF 2.0 QVT [qvt]). QVT is currently undergoing standardisation through the OMG, and a joint submission from the majority of the submitters is anticipated in November, 2004. We have chosen to use the concrete syntax from our submission in this paper because we have a prototype transformation engine, Tefkat [tefkat], that can evaluate this language, and as yet, there is no fixed concrete syntax for the new revised joint QVT model. Where we use the term 'QVT' in this paper, it should be interpreted as DSTC's QVT.

Mapping into property

UML has two model elements, *attribute* and *association*, which map to a single DL element, *property*. So if we wish to map further to ER, which has also two model elements *attribute* and *relationship*, we don't know which to choose unless we consult the source element. But UML, ER, DL and OWL all have the same underlying semantics, the theory of sets and relationships, so that there is no real semantic loss if a model is mapped from UML to ER via DL in such a way that all DL properties are mapped to ER relationships. The question becomes to see what the original modeler might have been doing in using both attributes and associations, so we can see what is lost.

From a software specification perspective the semantic content of a model is the externally visible behavior specified. If two representations of a specification have the same behavior specified, then each is a refinement of the other, so they are equivalent. So the identity of the semantics of the metamodels means that our mapping is an equivalent specification, even though it looks funny having all relationships and no attributes.

One reason UML modelers choose between attributes and associations is that the ultimate implementation is a relational database system where attributes are generally implemented as columns in a table while associations are implemented as multiple tables connected by foreign keys. The latter is much more expensive than the former, so the choice is an (implementation) engineering decision. Such implementation decisions are inappropriate in ontologies, so their loss in fact produces a better ontology.

However, another reason UML modelers choose between attributes and associations is that some connections contribute more to a human understanding of the model than others. So the more important connections are represented as associations so that the less important can be suppressed in overview renderings of the model. Loss of this distinction makes a difference to the understandability of the target model.

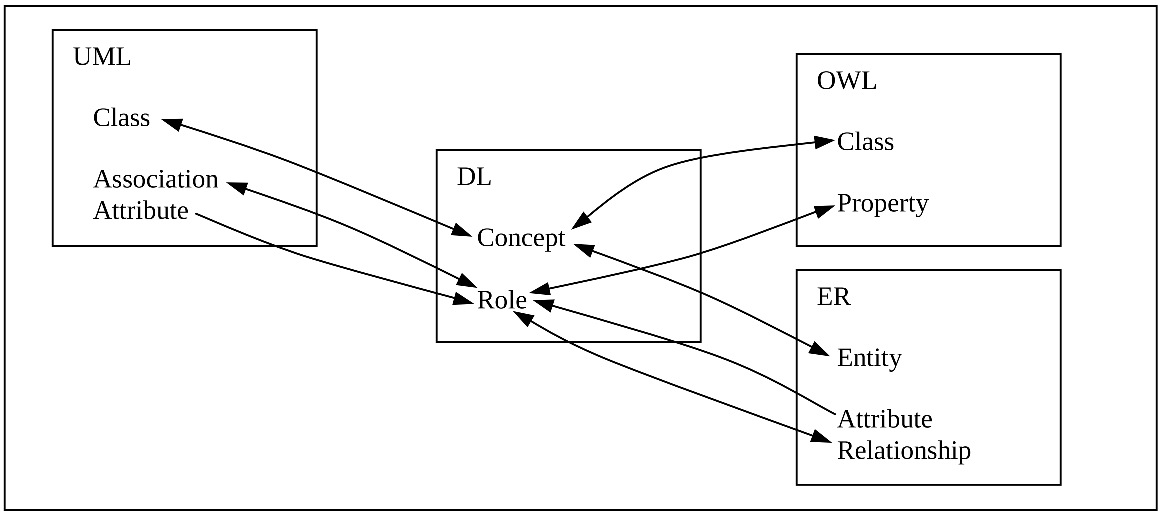

Figure 1: Naïve mappings between UML, DL, OWL and ER.

Using QVT, we constructed naive transformations between UML and DL, OWL and DL and ER and DL, as shown in Figure 1. This mapping produces models that are semantically equivalent, but which demonstrate structure loss. For example, while both UML Associations and Attributes are mapped to DL Roles when mapping from UML to DL, in the reverse, DL Roles are only ever mapped to UML Associations.

UML to DL

The rules that we use to map between UML and DL are shown in Figure 2. For brevity, we have elided details such as the import of the tracking model and some additional rules, for example, rules to map types of attributes. The rules Class2Concept and Concept2Class represent the mapping between Class and Concept (in both directions), and the rules Assoc2Role and Attr2Role represent mapping from UML Attributes and Associations to DL Roles, while Role2Assoc represents the naïve mapping from DL Role back to UML Association only.

```
TRANSFORMATION UML2DL(uml,dl)                TRANSFORMATION DL2UML(dl,uml)

RULE Assoc2Role (a,r)                        RULE Role2Assoc(r,a)
 FORALL UMLAssoc a                            FORALL DLRole r
 MAKE DLRole r                                MAKE UMLAssoc a
 SETTING r.uniqueID = a.name;                 SETTING a.name = r.uniqueID;

RULE Attr2Role (a,r)                         RULE Concept2Class(c1,c2)
 FORALL UMLAttr a                             FORALL DLConcept c1
 MAKE DLRole r                                MAKE UMLCLass c2
 SETTING r.uniqueID = a.name                  SETTING c2.name = c1.uniqueID;
 LINKING RoleForAttr
    WITH attr = a, role = r;

RULE Class2Concept(c1,c2)
 FORALL UMLClass c1
 MAKE DLConcept c2
 SETTING c2.uniqueID = c1.name;
```

Figure 2: QVT transformations between UML and DL

The LINKING statements in the rules identify the classes from the tracking model that are used to record correspondences (or trace information) between models when they are mapped. The tracking models for each of the transformations discussed in this paper are very basic. For example, the UML2DL tracking model used by the rules shown in Figure 2 contains the class RoleForAttr.

ER to DL

The mappings between ER and DL, as shown in Figure 3, are similar to the UML to DL rules: Entity and Concept and Relationship and Role are considered equivalent in the naïve mapping, from Figure 1, hence the rules Entity2Concept, Concept2Entity, Relationship2Role and Role2Relationship. ER Attributes map to DL Roles via the rule Attribute2Role.

```
TRANSFORMATION ER2DL(er,dl)                  TRANSFORMATION DL2ER(dl,er)

RULE Relationship2Role(r1,r2)                RULE Role2Relationship(r1,r2)
 FORALL ERRelationship r1                     FORALL DLRole r1
 MAKE DLRole r2                               MAKE ERRelationship r2
 SETTING r2.uniqueID = r1.name;               SETTING r2.name = r1.uniqueID;

RULE Attribute2Role(a,r)                     RULE Concept2Entity (c,e)
 FORALL ERAttribute a                         FORALL DLConcept c
 MAKE DLRole r                                MAKE EREntity e
 SETTING r.uniqueID = a.name;                 SETTING e.name = c.uniqueID;

RULE Entity2Concept(e,c)
 FORALL EREntity e
 MAKE DLConcept c
 SETTING c.uniqueID = e.name;
```

Figure 3: QVT transformations between ER and DL

OWL to DL

The naïve mapping between OWL and DL is basic; Class corresponds to Concept, and Property to Role. Figure 4 shows the QVT rules that represent these mappings.

Figure 4: QVT transformations between OWL and DL

```
TRANSFORMATION OWL2DL(owl,dl)

RULE Property2Role(p,r)
 FORALL OWLProperty p
 MAKE DLRole r
 SETTING r.uniqueID = p.localname,
 LINKING PropForRole
    WITH prop = p, role = r;

RULE Class2Concept(c1,c2)
 FORALL OWLClass c1
 MAKE DLConcept c2
 SETTING c2.uniqueID = c1.localname;
```

```
TRANSFORMATION DL2OWL(dl,owl)

RULE Role2Property(r,p)
 FORALL DLRole r
 MAKE OWLProperty p
 SETTING p.localname = r.uniqueID,
 LINKING PropForRole
    WITH prop = p, role = r;

RULE Concept2Class (c1,c2)
 FORALL DLConcept c1
 MAKE OWLClass c2
 SETTING c2.localname = c1.uniqueID;
```

Extending the mappings

To overcome structure loss, the naïve mappings can be extended with additional rules that make use of trace information over multiple transformations, for example, for mapping between UML and ER via DL. The rules can also be made more complex, to detect patterns in the source model that attempt to identify particular situations when the modeler deliberately used an attribute rather than an association or relationship, and to preserve this distinction in the target model.

Figure 5 shows an example of extending the mapping between DL and ER, so that if a Role was originally mapped from a UML Attribute, the Role is mapped to an ER Attribute instead of a Relationship.

Figure 5: Extended rule for ER attributes from UML

```
TRANSFORMATION DL2ER-fromUML(dl,uml2dltrace,er) EXTENDS DL2ER(dl,er)

RULE Role2Attribute-fromUML(r,a)
 SUPERSEDES Role2Relationship(r,r2)
 FORALL DLRole r, RoleForAttr@uml2dltrace ra
 WHERE ra.role = r
 MAKE ERAttribute a FROM r
 SETTING a.name = r.uniqueID;
```

In this example, we extend the DL2ER mapping, and then use QVT's rule superseding to override the rule Role2Relationship. The new transformation has an additional parameter; the trace model that was produced when the source UML model was mapped to DL. By looking up this trace, we can determine whether each Role was mapped from a UML Association or Attribute. (In this case, if there is a RoleForAttr class in the trace, the Role was mapped from a UML Attribute). Similar extensions could be written for the DL2UML mapping, for the reverse situation, to result in a UML Attribute for each DL Role that was mapped from an ER Attribute.

An advantage of this approach is that we can selectively apply the extended transformation only to those models where it is appropriate, while avoiding duplication of rules across many variations of the same transformation, as they all extend from a common set of rules. QVT's Rule superseding and extension allow us to change the behaviour of rules, or even to disable rules under certain conditions, as we see in the example shown in Figure 6, where we map a DL Role to a UML Attribute.

Figure 6: Extended rule for UML attributes from OWL

```
TRANSFORMATION DL2UML-fromOWL(dl,owl2dltrace,uml)
 EXTENDS DL2UML(dl,uml)

RULE Role2Attr(r,a)
 SUPERSEDES Role2Assoc(r,a)
 FORALL DLRole r, PropForRole@owl2dltrace pr
 WHERE pr.role = r AND pr.prop = p
  AND OWLDataTypeProperty p
 MAKE UMLAttribute a FROM r
 SETTING a.name = r.uniqueID;
  LINKING RoleForAttr
     WITH attr = a, role = r;
```

The addition of the WHERE clause allows for conditions to be applied. In this case, Role2Attr only replaces Role2Assoc when the Property that corresponds to the Role (stored in the tracking class PropForRole) is an OWLDataTypeProperty.

SCL is the full first-order predicate calculus, which none of the other metamodels support

SCL is a rich enough language that it could be used as the underlying semantics of OWL, and in fact is intended to be so used, so we can use QVT to map OWL to SCL via DL in the sort of way described in the previous section.

However, SCL is being put forward by the ODM working party as a predicate definition language in the RDF/RDFS/OWL metamodel. One way to do this is to specify a predicate *P* in OWL as a functional property whose range is {true, false}. The mapping from OWL to SCL makes the structures of the OWL ontology visible to an SCL engine. We can create a predicate in SCL which implements predicate *P*, but can't translate it into OWL because OWL lacks the necessary syntax. The image in OWL of the SCL implementation of P is simply the OWL predicate *P*. The mapping is lossy in the same way as the loss of the distinction between association and attribute in mapping from UML to DL.

The ability of QVT to keep track of the association between source and target objects is a link from the signature of the predicate (the OWL specification of *P*) and its implementation in SCL. An augmented OWL inference engine can use the tracking model to compute the predicate when it is needed.

This approach can apply to all the other metamodels. The predicates are named in the target model, using QVT to tie them to their SCL implementations. These target predicates can be further mapped to other metamodels, say from OWL to Topic Maps, with the ultimate source in SCL still being accessible via the QVT tracking model.

OWL Full classes as individuals

OWL Full allows classes to be individuals, therefore instances of other classes. In particular, an OWL Full model can represent its own metamodel, by for example mapping it from its MOF2 representation to OWL using QVT. First order systems generally and DL in particular do not allow this second-order representation.

However, since the underlying semantics is finite, there are equivalent first order representations of the second-order structures. For example, the individual class name could be reified as a class itself and attached to the ultimate individual as a property, resulting in a collection of subclasses of the union of the ultimate individuals indexed by the class of class names. But that representation makes the behavior much more difficult to specify, so loses refinement. It is also difficult to recover the original OWL Full model elements, since that is an example of the well-known problem of decompilation.

In this case, the QVT mapping from the OWL Full model to a first order model like DL will gain structure. It will be s sort of compilation of the OWL Full specification. So it is essentially the inverse of the SCL case described in the previous section. Modifications at the DL level break the inverse mapping, of course. The ability of QVT to associate the target with the source allows recovery of the OWL Full specification, so long as the DL representation is not independently modified.All of the relevant DL structure must be linked to the DL tracking model generated by the original mapping from OWL Full.

Conclusion

QVT is a mapping tool which does not simply translate a model from one metamodel to another, but keeps track of the association between source and target model elements. It can therefore be used to tie the ODM together, since its tracking model can support recoverable lossy mappings.

References

[odm] OMG *RFP for an Ontology Definition Metamodel* (ad/2003-03-40)
[qvt] CBOP, DSTC, IBM. *MOF Query/Views/Transformations, 2nd Revised Submission*. OMG ad/04-01-06.
[tefkat] *Tefkat Model transformation engine*. [Internet] Brisbane, Australia: DSTC. Available from <http://www.dstc.edu.au/Research/Projects/Pegamento/tefkat/index.html>