Transformations based on Relations

D.H.Akehurst

University of Kent
D.H.Akehurst@kent.ac.uk

The Model Driven Architecture (MDA) is an approach to IT systems development fostered by the Object Management Group (OMG). It is based on forming a separation between the specification of a systems essential functionality as a platform independent model (PIM) and the realisation of the system using more detailed and specific platform specification (PSM). It is recognised that specifying the mappings or transformations from PIM to PSM is a key enabling aspect of the MDA approach. Currently the OMG's Request for Proposals (RFP) on techniques and facilities to enable transformations is in progress. In this position paper we discus a technique for specifying transformations that is based on the mathematical foundation of relations. Using these relation specifications we show how the additional definition of some "build" expressions enables the generation of a transformation engine that will map model instances from either side of the specification to the other. This approach has been proved to work on a number of small case studies, using the KMF code generation tools to build transformation engines from specifications.

1 Specification Principles

The principle on which this transformation specification technique is based is that of mathematical relations. A Relation (R) is a set of elements, where each element is a pairing of two objects (a and b) that map to each other. The objects are each drawn from two other sets defined as the domain and range of the relation. An additional *matching condition* can be specified which is a relation specific expression that must evaluate to true for all elements in the relation. In mathematics this can be written:

$$R = \{(a,b) \in A \times B \mid a \in domain \wedge b \in range \wedge matching\_condition\}$$

In an Object Oriented modelling context we can express such a relation with a pattern of classes and constraints, as shown in Figure 1.

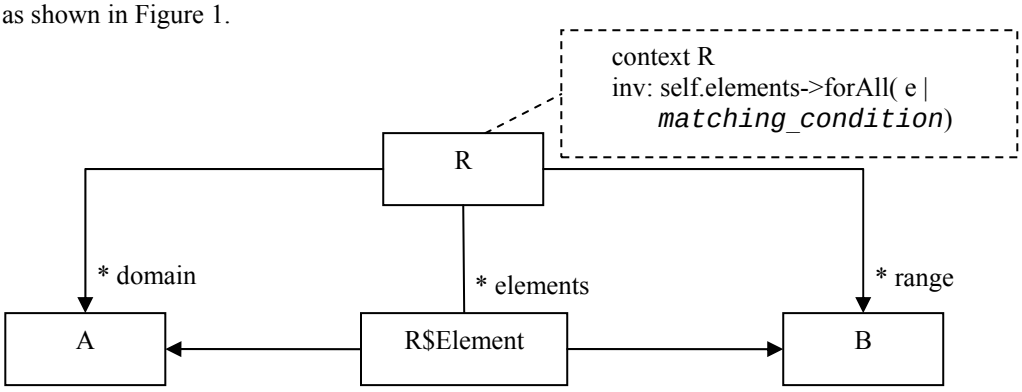

Figure 1

Given this pattern of classes we can define relations between classes from different models and specify the conditions under which the model instances should be mapped (i.e. contained in the relation). It is also necessary to define the domain and range sets; this can be 'allInstaces' of the domain and range types (classes) or we can define a more local 'scope' for each relation. One useful way to do this is by nesting relations into a hierarchy, using a subRelation relationship on which the domain and range expressions are defined for the sub relation. This causes the scope of one relation to be in the context of an element from its parent relation, and facilitates reuse and recursive nesting of relations; [2] discusses this in more detail.

To aid the concise specification of relations, we can use a single graphical icon to represent this pattern of classes or a single class can be used with a <<relation>> stereotype. These options are illustrated in Figure 2.

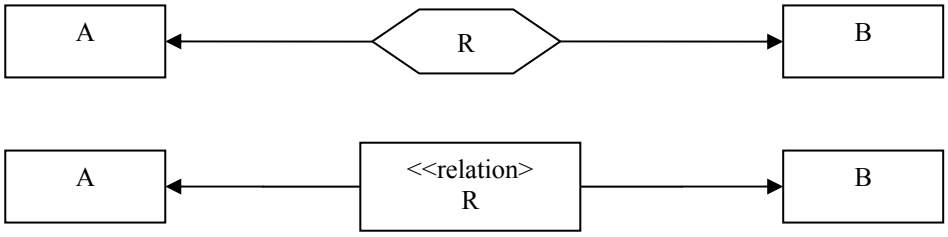

Figure 2

2 From UML to BPEL

We illustrate the specification technique with the example mapping between UML and BPEL. The Business Process Execution Language for Web Services (BPEL4WS or BPEL) is an XML based standard that provides a mechanism for combining Web Services to implement business processes. There is no graphical syntax defined for the language, consequently it proves useful to be able to define a process in a graphical notation such as UML Activity diagrams, and map these to a BPEL specification.

The document [4] describes a mapping between UML and BPEL, using this as a source, we show in the following figures parts of a transformation specification between the two metamodels. (The UML metamodel can be found in [5] and a segment of a metamodel for BPEL is shown in an appendix.)

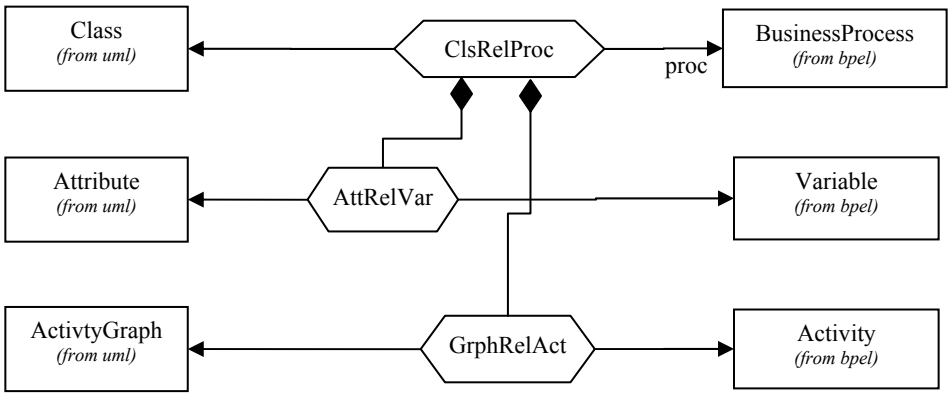

Figure 3 Mappings between UML and BPEL

The following OCL expressions define the domain, range and matching conditions for each of the relations specified in Figure 3.

```
context ClsRelProc$Element
  AttRelVar.domain :
    class.feature->select( f | f.oclIsKindOf(Attribute) )
  AttRelVar.range :
    proc.variable
  GrphRelAct.domain :
    class.behaviour->select( b | b.oclIsKindOf(ActivityGraph) )
  GrphRelAct.range :
    proc.activity
context AttRelVar$Element
  matchingCondition :
    attribute.name = variable.name and
    attribute.type.name = variable.type.name
context GrphRelAct$Element
  matchingCondition :
    true
```

The domain and range of both the Attribute<->Variable relation and the ActivityGraph<->Activity relation are defined within the scope of an owning relation element that maps a class to a business process.

The enclosing relation element gives a context in which to define the domain and range of its sub relations. This hierarchy of relations provides a tree of relations that map one model onto another, which (if changes are made to either side) can be used to update or check a transformation within local areas rather than having to check the whole transformation.

The matching conditions are defined in the context of relation elements, if these conditions do not evaluate to true, then the mapping is invalid and should be updated.

3 Tool Support

Currently, we use a generic UML class diagramming tool to draw the relations and annotate them with OCL expressions for the matching conditions. These models are imported by the KMF tools [3] to generate transformers, i.e., code for performing the transformations. These transformers need to know how to build objects from one side of the relation given objects from the other side. A constraint solver of some description could potentially do this for simple matching conditions. However there are limitations to what a constraint solver can do and they are generally pretty inefficient. Our alternative approach is to explicitly define expressions for each relation that will build an object of one side given an appropriate object from the other. The result of such an expression should give a new object which when paired with the original meets the matching condition.

Each build expression is interpreted in the context of the relation element that maps a domain element onto a range element. Depending on the direction of transformation, domain→range or range→domain, the appropriate domain or range build expression is used to build a missing element. After evaluating the build expression the matching condition must evaluate to true.

The following expressions illustrate the definition of a build expression for the Attribute<->Variable relation, using an action language built on top of OCL:

```
context AttRelVar$Element
  buildDomainExpression:
    uml::Attribute{ name = variable.name,
                    type = Class.allInstances->any( t |
                             t.name = variable.type.name) }
  buildRangeExpression:
    bpel::Variable{ name = attribute.name,
                    type = Type.allInstances->any(t |
                             t.name = attribute.type.name) }
```

These expressions build an instance of the corresponding class defined in the relation, then set the properties of that object so that the matching condition will evaluate to true. In both directions, the name property can be simply copied from the opposite objects name, however the type property is more complex. In this example we have assumed the existence of the 'allInstances' collection on model classes and used this to select a corresponding type. An alternative might be to explicitly build an appropriate object for the type, or if the transformation specification as a whole carries mappings between the types defined in each model we could look up an object in the transformation mappings.

Depending on the complexity of the relations and matching conditions in a transformation specification, the build expressions can be very complicated, however, by localising the scope of the expression to the context of a single relation the complexity is manageable.

4 Conclusion and Future Work

We are currently trying out this technique on a variety of different example, looking for difficult transformation problems and finding the limitations of the relation based approach to specifying transformations.

We find that the ability to localize the scope of a relation (defining the domain and range of sub-relations in the context of a parent relation element) is of much use with respect to managing the complexity; and is in particular useful with respect to the reuse of relations when we have recursive structures in one or other model, for example the UML package within packages structure.

One transformation problem we have not currently succeeded in specifying as a set of relations is that of a parser. Parser grammars for textual languages can be seen as transformations from a sequence of characters (or tokens) into an abstract syntax tree; the difficulties arise with respect to syntax that involves constructs such as nested parentheses, as often found in expression languages.

On the other hand we have succeeded in using the technique for specifying parsers for simple graphical languages such as Class or State Diagrams [1, 2], and have recently investigated the use of the technique for defining viewpoints [6]

Bibliography

[1] Akehurst D. H., "An OO Visual Language Definition Approach Supporting Multiple Views," in proceedings VL2000, IEEE Symposium on Visual Languages, September 2000.

[2] Akehurst D. H., Kent S., and Patrascoiu O., "A relational approach to defining and implementing transformations between metamodels," *Journal on Software and Systems Modeling*, vol. 2, pp. 215, November 2003.

[3] KMF-team, "Kent Modelling Framework (KMF)," 2002, www.cs.kent.ac.uk/projects/kmf

[4] Mantell K., "From UML to BPEL," 2003, http://www-106.ibm.com/developerworks/webservices/library/ws-uml2bpel/

[5] OMG, "The Unified Modeling Language Version 1.5," Object Management Group, formal/03-03-01, March 2003.

[6] Steen M. W. A., Doest H. L. t., Lankhorst M. M., and Akehurst D. H., "Supporting Viewpoint-Oriented Enterprise Architecture," in proceedings EDOC 2004, submitted.

Appendix – Segment of a Metamodel for the BPEL Language