Using the metamodel matching and transformation synthesis tools

K. Lano

February 15, 2020

1 Introduction

The Agile UML toolset provides techniques for deducing matchings of classes and features between metamodels. These matchings can be used to derive model transformations in UML-RSDS, ATL, ETL, QVT-O and QVT-R.

The latest version of the tools can be obtained from: https://nms.kcl.ac.uk/kevin.lano/uml2web/. These form part of the Agile UML toolset (https://projects.eclipse.org/projects/modeling.agileuml).

2 Metamodel matching

Metamodels should be loaded using the File menu option *Recent* (this loads the file output/mm.txt) or *Load metamodel*. Classes in the metamodel(s) should be marked as *source*, ie., with this stereotype, if they are in the source metamodel of the matching, and as *target* if they are in the target metamodel. Unmarked classes are assumed to be shared (in both metamodels and mapped to themselves).

As an example, Figure 1 shows the metamodels of the ATL Class2Relational transformation case from the ATL zoo (www.eclipse.org/atl/atlTransformations). The source metamodel MM_1 is *Class*, on the LHS, the target metamodel MM_2 is *Relational*, on the RHS.

In KM3 format the source metamodel is written as:

```
package Class {

abstract class NamedElt {
attribute name : String;
}

abstract class Classifier extends NamedElt {
}
```

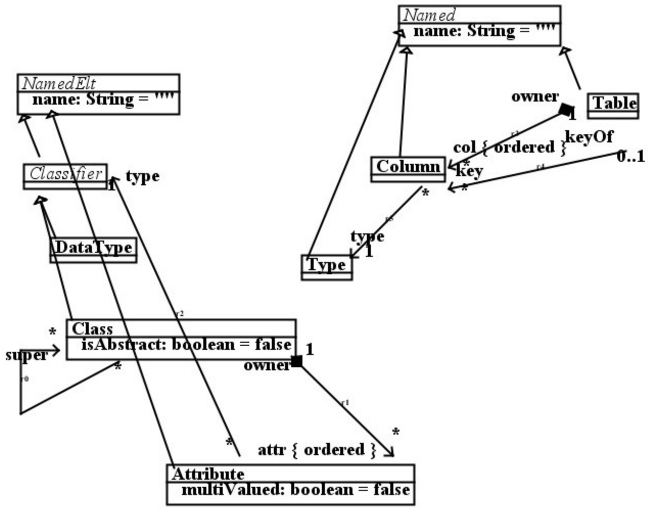

Figure 1: Class and Relational metamodels

```
class DataType extends Classifier {

}

class Class extends Classifier {
reference super[*] : Class;
reference attr[*] ordered container : Attribute oppositeOf owner;
attribute isAbstract : boolean;
}

class Attribute extends NamedElt {
attribute multiValued : boolean;
reference type : Classifier;
reference owner : Class oppositeOf attr;
}
}
```

The target metamodel is:

```
package Relational {

abstract class Named {
attribute name : String;
}

class Table extends Named {
```

```
reference col[*] ordered container : Column oppositeOf owner;
reference key[*] : Column oppositeOf keyOf;
}

class Column extends Named {
reference owner : Table oppositeOf col;
reference keyOf[0-1] : Table oppositeOf key;
reference type : Type;
}

class Type extends Named {

}
}
```

Once both metamodels are loaded, select the option *Synthesise transformation* from the *Synthesis* menu. This provides several options for matching strategies (Table 1). A matching comprises a relation *cm* between the classes of the two metamodels, and a relation *fm* between the features.

Measure	*Definition*
Data structure similarity (DSS)	Classes possess similar data in their owned, inherited or composed features [2]
Graph structural similarity (GSS)	Class neighbourhoods in the 2 metamodels have similar graph structure metrics [7]
Graph edit similarity (GES)	Class reachability graphs in the 2 metamodels have low graph edit distance [1]
Name syntactic similarity (NSS)	Classes have names with low string edit distances [6]
Name semantic similarity (NMS)	Class names are synonymous terms or in the same/linked term families according to a thesaurus [3]
Semantic context similarity (SCS)	Classes play similar semantic roles in the 2 metamodels [8].

Table 1: Syntactic and semantic similarity measures for classes

For DSS either a general matching can be used, or matchings can be restricted to be *inheritance preserving*: a subclass D of source class C is only permitted to map to a class $C1$ which C maps to, or to a subclass/descendant of such a $C1$.

For small examples such as the class/relational mapping, the NMS or DSS options are suitable. NMS uses a thesaurus (in output/thesaurus.txt) to match classes, so it is more appropriate if there are some linguistic similarities between the metamodels (such as *NamedElt* and *Named*, or *Class* and *Table*). If the metamodels have quite different terminologies then DSS is more suitable.

The tool will prompt you for the maximum navigation path to be considered on the source and target side. This means the maximum length of feature chains such as $super.isAbstract$ or $key.type$ (both of length 2). For cases where there is a close structural similarity between the metamodels, the choice of length 1 for source and target is usually adequate.

The results of the matching are shown in the console (Figure 2) and written to $output/forward.tl$ for the forward mapping, and $output/reverse.tl$ for the reverse mapping.

For example, the initial forward matchings derived by NMS with maximum source and target navigation length 1 look as follows:

$$
\begin{array}{l}
\mathit{NamedElt} \longmapsto \mathit{Named} \\
\quad \mathit{name} \longmapsto \mathit{name} \\
\mathit{Class} \longmapsto \mathit{Table} \\
\quad \mathit{name} \longmapsto \mathit{name} \\
\quad \mathit{attr} \longmapsto \mathit{col} \\
\mathit{Attribute} \longmapsto \mathit{Column} \\
\quad \mathit{name} \longmapsto \mathit{name} \\
\quad \mathit{owner} \longmapsto \mathit{owner} \\
\quad \mathit{type} \longmapsto \mathit{type} \\
\mathit{Classifier} \longmapsto \mathit{Type} \\
\quad \mathit{name} \longmapsto \mathit{name} \\
\mathit{DataType} \longmapsto \mathit{Type} \\
\quad \mathit{name} \longmapsto \mathit{name}
\end{array}
$$

However, this matching is incomplete on both target and source sides (*isAbstract*, *multiValued* and *super* are unused source features, *key* and *keyOf* are unused target features). In addition, there is a potential inconsistency in that *Class* is mapped to *Table*, but *Table* is not a specialisation of (or equal to) the image *Type* of *Classifier*, even though *Class* is a specialisation of *Classifier*.

An interactive process following the matching derivation is used to identify such flaws and to suggest possible resolutions.

Table 2 summarises the different checks which we use.

For the case of feature mapping incompleteness in Class2Relational, because the unused source feature *super* is a self-association on *Class*, the system proposes to replace $attr \longmapsto col$ by the mapping

$$
\begin{array}{l}
\mathit{Set}\{\mathit{self}\}{\rightarrow}\mathit{closure}(\\
\qquad \mathit{super}){\rightarrow}\mathit{unionAll}(\mathit{attr}) \;\; \longmapsto \;\; \mathit{col}
\end{array}
$$

of all defined attributes of a class to the columns of a table, ie., all attributes of the class itself and of all its ancestors are mapped to columns of the table corresponding to the class (Figure 2).

Because of the inheritance conflict in the targets of the class mappings, the

Issue	*Correction*
Class mapping $Sub \longmapsto T$ for Sub subclass of E, has T not subclass/or equal to F, where $E \longmapsto F$	Retarget Sub mapping, or add target splitting map $Sub \longmapsto F$
Two directions of bidirectional association r not mapped to mutually reverse target features	Modify one feature mapping to ensure consistency
Source, target features have different multiplicities	Propose modified mappings
Unused target subclasses $F1$ of F, where $E \longmapsto F$	Introduce condition $F1C$ and mapping $\{F1C\}E \longmapsto F1$
Unused source or target feature f	Suggest class or feature mapping that uses f
Feature mapping $f \longmapsto r.g$ with $r : R$ of abstract type/element type	Propose concrete subclass $RSub$ of R for instantiation of r.

Table 2: Consistency and completeness checks

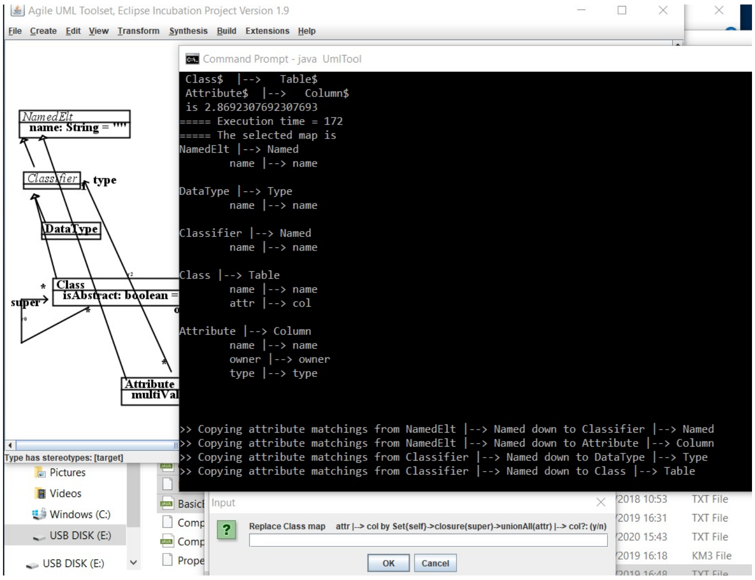

Figure 2: Initial metamodel matching

additional class mapping

$$\begin{aligned} &\mathit{Class} \longmapsto \mathit{Type} \\ &\quad \mathit{name} \longmapsto \mathit{name} \end{aligned}$$

is also proposed: this is a 'vertical class splitting' of *Class*: each *Class* instance in a source model is represented by both a *Type* instance and a *Table* instance in the resulting *Relational* model[1].

In the final stage of metamodel matching, the details of the matching and any correspondence patterns identified are listed in the console (Figure 3). Warnings are given in cases (such as multiplicity or type narrowing of target features relative to the source) where semantic problems may arise in mapping source models to target models.

3 Generating transformation specifications

Together with the metamodel matchings, the tool produces files *forward.txt* and *reverse.txt*, which contain transformation specifications for the two directions of the matching, in QVT-R, QVT-O, UML-RSDS, ATL and ETL.

While class matchings translate to rules in the MT languages, sometimes multiple class matchings must be combined in a single rule (in ATL), or one class matching is split into several rules (in QVT-R). In ATL and ETL, composite

[1]The target classes must have no common MM_2 ancestor which is a type/element type of some $g \in \mathrm{ran}(fm)$

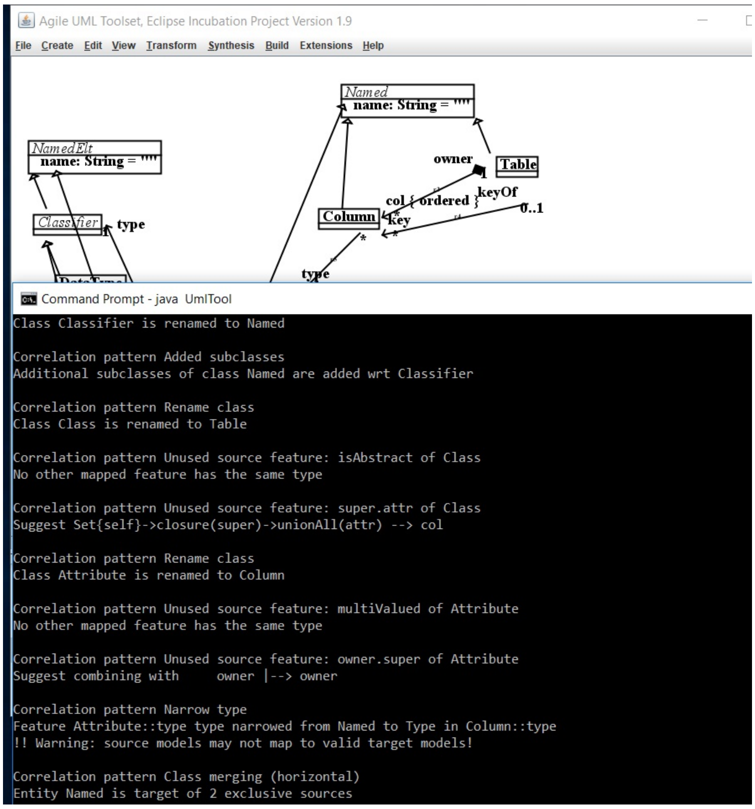

Figure 3: Metamodel matching with correspondence patterns

target features in mappings $f \longmapsto g.h$ must be implemented using additional lazy/called rules. In QVT-R, QVT-O and ETL rule inheritance is used to remove redundant mappings (in cases where the same feature mappings occur for a class and its superclass).

For example, the synthesised QVT-R of the above case is:

```
transformation tau(source: MM1, target: MM2)
{
  abstract top relation NamedElt2Named
  { checkonly domain source namedelt$x : NamedElt {};
    enforce domain target named$x : Named {};
  }

  abstract top relation Classifier2Type overrides NamedElt2Named
  { checkonly domain source classifier$x : Classifier {};
    enforce domain target type$x : Type {};
  }

  top relation DataType2Type overrides Classifier2Type
  { checkonly domain source datatype$x : DataType {};
    enforce domain target type$x : Type {};
  }

  top relation Class2Table overrides Classifier2Type
  { checkonly domain source class$x : Class {};
    enforce domain target table$x : Table {};
  }

  top relation Attribute2Column overrides NamedElt2Named
  { checkonly domain source attribute$x : Attribute {};
    enforce domain target column$x : Column {};
  }

  top relation Class2Type overrides Classifier2Type
  { checkonly domain source classx : Class {};
    enforce domain target typex : Type {};
  }

  top relation MapDataType2Type
  {  checkonly domain source
    datatype$x : DataType { name = datatype$x_name$value };
    enforce domain target
    type$x : Type { name = datatype$x_name$value, typeFlag = "DataType" };
    when {
    DataType2Type(datatype$x,type$x) }
  }
```

```
  top relation MapClass2Table
  {   domain source var$0 : Attribute {};
   checkonly domain source
    class$x : Class { name = class$x_name$value }
    { Set{class$x}->closure(super)->unionAll(attr)->includes(var$0)  };
    enforce domain target
    table$x : Table { col = table$x_col$x : Column {  },
 name = class$x_name$value };
    when {
    Class2Table(class$x,table$x) and
           Attribute2Column(var$0,table$x_col$x) }
  }

  top relation MapAttribute2Column
  {
 checkonly domain source
    attribute$x : Attribute { name = attribute$x_name$value,
      owner = attribute$x_owner$x : Class {  },
      type = attribute$x_type$x : Classifier {  } };
    enforce domain target
    column$x : Column { name = attribute$x_name$value,
      owner = column$x_owner$x : Table {  },
      type = column$x_type$x : Type {  } };
    when {
    Attribute2Column(attribute$x,column$x) and
                Class2Table(attribute$x_owner$x,column$x_owner$x) and
           Classifier2Type(attribute$x_type$x,column$x_type$x) }
  }

  top relation MapClass2Type
  {  checkonly domain source
    classx : Class { name = classx_name$value };
    enforce domain target
    typex : Type { name = classx_name$value };
    when {
    Class2Type(classx,typex) }
  }

}
```

References

[1] H. Bunke, K. Riesen, *Recent advances in graph-based pattern recognition*, Pattern Recognition 44, pp. 1057–1067, 2011.

[2] S. Fang, K. Lano, *Extracting Correspondences from Metamodels Using Metamodel Matching*, Doctorial Symposium, STAF 2019.

[3] D. Kless, S. Milton, *Comparison of thesauri and ontologies from a semiotic perspective*, AOW 2010.

[4] K. Lano, S. Kolahdouz-Rahimi, M. Sharbaf, H. Alfraihi, *Technical debt in Model Transformation specifications*, ICMT 2018.

[5] K. Lano, S. Fang, *Automated synthesis of ATL transformations from metamodel correspondences*, Modelsward 2020.

[6] I. Levenshtein, *Binary codes capable of correcting deletions, insertions and reversals*, Cybernetics and control theory, 10(8), 1966, pp. 707–710.

[7] O. Macindoe, W. Richards, *Graph comparison using fine structure analysis*, IEEE 2nd Int. Conf. Social Computing, 2010.

[8] A. Maedche, S. Staab, *Comparing ontologies – similarity measures and a comparison study*, EKAW 2002.

[9] S. Melnik, H. Garcia-Molina, E. Rahm, *Similarity flooding: a versatile graph matching algorithm and its application to schema matching*, 18th International Conf. Data Engineering, IEEE, 2002, pp. 117–128.

[10] MetamodelRefactoring.org, *Metamodel refactorings catalog*, www.metamodelrefactoring.org, 2020.

[11] OMG, *MOF2 Query/View/Transformation v1.3*, 2016.