Types Extraction and Similarity Matrixes in ATL Transformations – Technical Report

Javier Troya[1], Loli Burgueño[2], Manuel Wimmer[1], and Antonio Vallecillo[2]

[1] Business Informatics Group, Vienna University of Technology, Austria
{troya,wimmer}@big.tuwien.ac.at
[2] Dpto Lenguajes y Ciencias de la Computación, Universidad de Málaga, Spain
{loli,av}@lcc.uma.es

Abstract. In this technical report we explain with detail the types extraction in ATL transformations. Since there is no current support by ATL in realizing this task, it acquires a certain degree of difficulty. Also, we show how we use the types extraction to obtain the so-called similarity matrixes.

1 Introduction

The procedure described in this document has been deployed in order to implement the approach presented in [1]. In such work, a light-weight approach to automatically checking model transformations is presented. It is based on matching functions that establish alignments between specifications and implementations using the metamodel footprints, i.e., the metamodel elements used. The approach is implemented for the combination of Tracts and ATL, both residing in the Eclipse Modeling Framework, and is supported by the corresponding toolkit.

Tracts [4,3] are expressed in OCL (Object Constraint Language). For this reason, the API of USE (UML based Specification Environment) tool [2] can be used in the types extraction, what eases the process. Contrarily, ATL does not offer any support nor API to do the extraction, what makes it quite complex. The way we have addressed it is expressed in the next section of this technical report.

For obtaining the fitness of a model transformation, this is, how accurate it will be to apply the approach on such transformation, we define the concept of *similarity matrix*. A similarity matrix gives us how related rules are among them, i.e., the factor of common types they have. In this technical paper we also show the results of these matrixes when applied over a large number of model transformation taken from the ATL zoo[3].

2 Types Extraction

The first step in the types extraction is to inject the textual ATL transformation into a model-based representation. It is done automatically by means of a text-

[3] `http://www.eclipse.org/atl/atlTransformations/`

to-model transformation. The obtained model conforms to the ATL metamodel, which is in turn made up of three packages: ATL, OCL and PrimitiveTypes. Then, an ATL transformation (in fact, a so-called higher-order transformation) takes the obtained model, as well as the input and target metamodels of the original transformation, and generates a model with information of the types used in each and every rule. Focusing on a rule (let us say a matched rule for instance), it is quite straightforward to obtain the types of the elements in the left-hand side (LHS, the input part) of the rules as well as those created in the righ-hand side (RHS, the output part). To do so, we need to navigate those objects of type InPattern, OutPattern and Binding of the ATL package[4]. The most challenging part is to extract the types from the OCL expressions, which can be present in the filter part (of the LHS), local variables, the RHS and the imperative part. These textual expressions are built conforming to the OCL package[5] of the ATL metamodel. The extraction of the types in the OCL expressions is a three-step process. In the first step, we only need information of the ATL transformation (expressed as a model, as explained before), while in the second and third steps we need information of the source and target metamodels of the transformation in order to be able to navigate them.

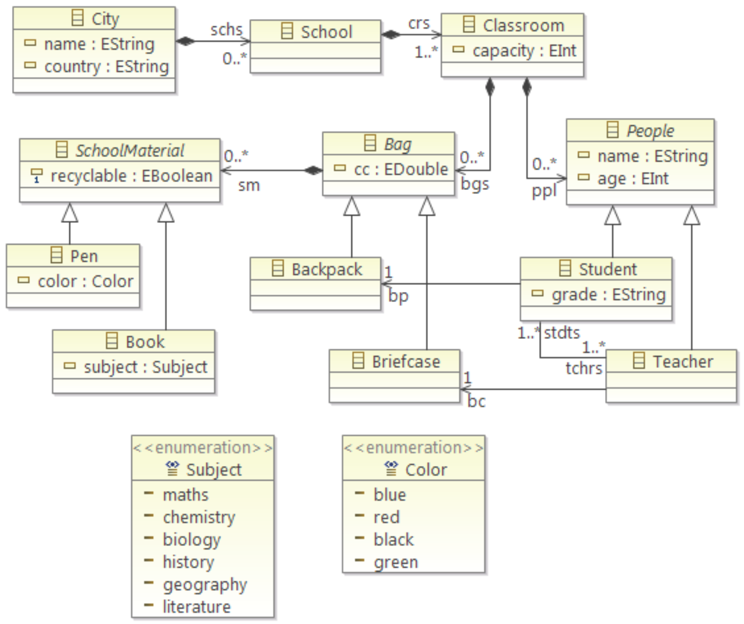

Fig. 1: Metamodel for organizing schools

[4] A snapshot of the ATL package is available from `http://atenea.lcc.uma.es/Descargas/ATL.png` (the references to the OCL package are not displayed)

[5] A snapshot of the OCL package is available from `http://atenea.lcc.uma.es/Descargas/OCL.png` (the references to the ATL package are not displayed)

An OCL expression can be made up of iterators (in a model level, they are objects of type IteratorExp), such us collect and select. The first step of the types extraction consists of taking every OCL expression and removing the iterators. When doing so, from each OCL expression (that may contain iterators), one or more navigation paths are obtained. Let us explain this with an example. Considering the metamodel of Figure 1, imagine we have within a transformation the following OCL expression: City.allInstances() -> collect(c | c.schs.crs) -> select(c | c.capacity>15) -> collect (ppl.name). First of all, let us recall the purpose of an OCL expression in a model transformation: it is to retrieve an object, or a collection of objects, that need to be accessed by means of a navigation through other elements in the model. In OCL, this is expressed by navigating through the references in a metamodel level. The collect operation is used to specify a collection that is derived from some other collection, and which contains different objects from the original collection. When we have more than one collect operation in an expression, we want to navigate until the last feature of the last collect. If a select or reject operation is found between two collect operations, we ignore it because it does not add information about the overall navigation path. Consequently, for the given OCL expression, one of the navigation paths extracted is City.schs.crs.ppl.name. Although we have ignored the select operation in this navigation path, it has an influence in the object (or collection or objects) extracted by the expression because, at some point in the navigation, it selects a subset of a collection. This is why, for each select operation, we also extract a path that goes from the beginning of the OCL expression until it. Consequently, in the given example, we also extract City.schs.crs.capacity.

With the first step, we manage to transform the OCL expressions that contain iterators into navigation paths from a class in a metamodel until a given feature. Said paths navigate through references in the metamodel. In the second step, we substitute the references in the navigation path for the type (the target class) of such references. The last element in the path, which is an object of type EStructuralFeature according to the Ecore metamodel[6], remains as it is. For the first navigation path extracted before, the path obtained now is City.School.Classroom.People.name. In order to create it, we need to access the metamodel (Figure 1) for obtaining the target class (eType) of each reference. An OCL expression may also navigate through the reference that starts from a subclass of a certain class. For example, the OCL expression School.allInstances() -> collect(s | s.crs.ppl) -> select(p | p.age < 16) -> collect(p | p.bp) -> first().cc is accessing the reference bp from objects of abstract type People. This means that the objects are, in fact, of type Student. It is important to take this into consideration in this step. The navigation path extracted, in the first step, from this expression is: School.crs.ppl.bp.cc; and the path extracted in the second step is: School.Classroom.People.Backpack.cc. Similarly, an OCL expression may navigate through the reference starting from a superclass of a certain class. Consider for example the OCL expression Classroom.allInstances()

[6] A snapshot of the Ecore metamodel is available from `http://atenea.lcc.uma.es/Descargas/Ecore.png`

-> collect(c | c.ppl.bc) -> forAll(b | b.sm.recyclabe). In the first step, we get the navigation path Classroom.ppl.bc.sm. For the second step, we have to know that sm reference departs from a superclass of Briefcase. We obtain the path Classroom.People.Briefcase.SchoolMaterial.recyclable. The last OCL expression shown contains a forAll operation. It often appears in the filter part of ATL rules and at the last part of the expression, so we navigate until it.

The third step consists of chopping the path obtained in the second step, so that we finally have the most significant types from the OCL expression. As explained in [1], we only leave the last two features. Thus, for path School.Classroom. People.Backpack.Bag.cc, we only take Bag.cc.

Finally, and as also described in [1], we return as output the type(s) of the last expression. For the case of Bag.cc, we return Bag.cc and Bag, since the type of cc is a primitive type. If the last feature is a reference instead of an attribute of a primitive type, we return its type. For instance, from the OCL expression Student.allInstances() -> select(s | s.grade = '4A') -> collect(s | s.bp.sm), one of the navigation paths we get is (first step) Student.bp.sm, from which we obtain (second step) Student.BackPack.sm, then we chop it and have (third step) Backpack.sm. Finally, we return Backpack.sm, Backpack and SchoolMaterial. When the last feature is of an enumeration type, we retrieve such type. For example, if we obtain in the third step Book.subject, then we return Book.subject, Book and Subject.

3 Similarity Matrix

As explained in [1], we use similarity matrixes to obtain the *fitness* of a given model transformation. They give us how related rules are among them, i.e., the factor of common types they have. These matrixes, having rules in both columns and rows, are consequently symmetric. Thus, we are only interested in half the matrix (excluding the main diagonal, of course). To calculate the fitness for the transformation, we extract the average and standard deviation of half the table. The lower both values are, the fewer types rules have in common, and the fitter a transformation is for applying our approach. We do not only use the arithmetic mean, but also the standard deviation, because we need all values to be low.

We recommend to apply our approach on model transformations where the mean and standard deviation of the similarity matrix for rules are below 0.15. Otherwise, the accuracy of the results is not good enough (precision would be normally below 0.7). For example, the similarity matrix for the Ecore2USE case study[7] gives a standard deviation of 0.18, and the precision after applying our approach is 0.6 [1]. On the other hand, for the CPL2SPL case study[8], the mean of the similarity matrix is 0.08, while the standard deviation is 0.1. In this case, the precision of the results is 0.8.

We have applied our ATL types extractor and have calculated the similarity matrix of 41 model transformations taken from the ATL zoo. Out of those, it is advisable to apply our approach on 21 of them, while it is not in the rest, as

[7] http://atenea.lcc.uma.es/index.php/Main_Page/Resources/MTB/Ecore2USE
[8] http://atenea.lcc.uma.es/index.php/Main_Page/Resources/MTB/CPL2SPL

shown in Table 1. The similarity matrixes of all of them, as well as their mean and standard deviation, are available on our website[9].

Transformation	# Rules	Mean	Deviation
🟥 **ATL2Problem**	18	0.35	0.13
🟩 **ATOM2RSS**	3	0.05	0.05
🟥 **ATOM2XML**	8	0.33	0.08
🟥 **BibTex2DocBook**	9	0.41	0.24
🟩 **CPL2SPL**	16	0.07	0.13
🟩 **Ecore2USE**	14	0.1	0.17
🟩 **Grafcet2PetriNet**	5	0.14	0.07
🟥 **HTML2XML**	30	0.21	0.11
🟩 **IEEE14712MoDAF**	13	0.03	0.05
🟩 **KM32OWL**	16	0.13	0.14
🟥 **KM32Problem**	16	0.44	0.15
🟥 **Measure2Table**	6	0.31	0.37
🟩 **Measure2XHTML**	22	0.07	0.11
🟥 **MySQL2KM3**	11	0.2	0.29
🟩 **OCL2R2ML**	37	0.09	0.12
🟥 **OWL2XML**	24	0.51	0.16
🟩 **PathExp2PetriNet**	3	0.15	0.04
🟥 **PathExp2TextualPath**	5	0.37	0.44
🟩 **PetriNet2Grafcet**	5	0.14	0.07
🟥 **PetriNet2PathExp**	3	0.28	0.11
🟥 **PetriNet2PNML**	4	0.17	0.05
🟥 **PetriNet2XML**	5	0.54	0.11
🟥 **PNML2PetriNet**	5	0.28	0.12
🟥 **PNML2XML**	4	0.72	0.17
🟩 **R2ML2RDM**	69	0.11	0.14
🟥 **R2ML2XML**	55	0.26	0.13
🟩 **R2ML2WSDL**	14	0.07	0.14
🟩 **RDM2R2ML**	56	0.1	0.13
🟥 **RDM2XML**	39	0.32	0.14
🟩 **RSS2ATOM**	3	0.05	0.05
🟥 **RSS2XML**	4	0.37	0.15
🟩 **UML2ER**	8	0.09	0.11
🟩 **WSDL2R2ML**	17	0.06	0.11
🟥 **WSDL2XML**	20	0.36	0.15
🟩 **XML2ATOM**	10	0.15	0.06
🟩 **XML2MySQL**	6	0.12	0.1
🟥 **XML2PetriNet**	5	0.29	0.06
🟥 **XML2PNML**	5	0.25	0.19
🟩 **XML2RSS**	9	0.14	0.07
🟩 **XML2WSDL**	19	0.14	0.08
🟩 **XSLT2XQuery**	7	0.07	0.14

Table 1: Summary of Similarity Matrixes. Green: those where the approach is advisable to be applied. Red: those where it is not

We discovered that the number of rules in the transformations has no impact in the accuracy of our approach. In fact, the number of rules used in the set of transformations studied ranged from 3 up to 69. As an example, the similarity matrix of a small transformation (*PetriNet2PathExp*, 3 rules) gave bad

[9] `http://atenea.lcc.uma.es/index.php/Main_Page/Resources/MTB/SimilarityMatrix`

results, while the one obtained from the largest transformation (*R2ML2RDM*, 69 rules) gave good results. Contrarily, we obtained adverse results for another large transformation (*R2ML2XML*, 55 rules), while we got good results for small transformations (such as *PetriNet2Grafcet*, 5 rules). We have applied the *Pearson product-moment correlation coefficient*, a measure of the linear correlation (dependence) between two variables, on the results, when the first variable is the number of rules in the transformations and the second is the mean obtained from the similarity matrixes. The value of this measure varies in the range $[-1, 1]$, where 1 is total positive correlation, 0 is no correlation, and -1 is negative correlation. The results we obtained is -0.13, meaning that this dependence is minimal.

4 Conclusion

In this technical report we have explained in detail with a running example how we manage to extract the types in an ATL transformation. This implementation is part of our work [1] to automatically checking model transformations based on matching functions that establish alignments between specifications and implementations using the metamodel footprints, i.e., the types used in transformations and constraints. Furthermore, we have shown the approach we use to check before hand if the accuracy of our proposal [1] will be fair for a given model transformations, namely similarity matrixes. We conclude that the applicability of our proposal does not depend on how large model transformations are according to their number of rules.

References

1. Burgueno, L., Troya, J., Wimmer, M., Vallecillo, A.: Checking model transformations using tracts. Transaction on Software Engineering (submitted for publication) (2013)
2. Richters, M., Gogolla, M.: OCL: Syntax, semantics, and tools. In: Object Modeling with the OCL (2002)
3. Vallecillo, A., Gogolla, M.: Typing model transformations using tracts. In: Hu, Z., Lara, J. (eds.) Theory and Practice of Model Transformations, Lecture Notes in Computer Science, vol. 7307, pp. 56–71. Springer Berlin Heidelberg (2012), `http://dx.doi.org/10.1007/978-3-642-30476-7_4`
4. Wimmer, M., Burgueño, L.: Testing M2T/T2M transformations. In: Proc. of the 16th International Conference on Model-Driven Engineering Languages and Systems. pp. 203–219. LNCS 8107 (2013)