Using ATL in a tool-chain to calculate coverage data for UML class diagrams

- Short Paper -

Hao Wu*, Rosemary Monahan, and James F. Power

Department of Computer Science, National University of Ireland, Maynooth
{haowu,rosemary,jpower}@cs.nuim.ie

Abstract. In this paper we describe the use of ATL as part of a tool chain that calculates coverage measures for UML class diagrams. The tool chains uses the USE tool as a parser and validator for UML diagrams, and represents the diagrams internally using the EMF framework.

1 Introduction and Related Work

Typically, test suites for source code can be measured in terms of their *coverage* of code features, such as statement, condition or decision coverage. In the context of Model Driven Engineering (MDE), a considerable body of research exists in the area of model-based testing [1, 2], and a number of coverage criteria have been proposed for UML diagrams [3–7]. Such criteria can be used not only in the evaluation of an existing test suite but also to guide the automatic generation of test cases [5].

In this paper we restrict our attention to structural specification, in particular UML class diagrams. These have the unusual feature that their instances, namely object diagrams, are explicitly available in UML, making it possible to calculate coverage data entirely within the UML framework. In this context, coverage calculation becomes a matter of quantifying these diagrams, and is thus closely related to deriving metrics for models. While there have been a number of different approaches to this problem, we limit ourselves to those based entirely in an MDE framework, and distinguish three main *types* of approach:

I. Define the metrics directly as queries over the UML diagram [8].
II. Define a transformation from the UML diagram to a measurement metamodel [9, 10]
III. Define a transformation from the UML diagram to a "measurable entities" metamodel, and define the metrics as queries over this metamodel [11].

Of course many other variations are possible. A further elaboration of approach III, not considered here, would be to represent the metric specifications themselves explicitly as a metamodel, following the UML22Measure approach [12].

* This work is supported by a John & Pat Hume Scholarship from NUI Maynooth.

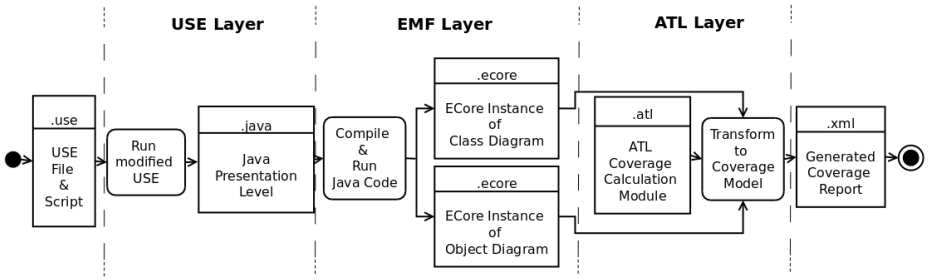

Fig. 1. This UML activity diagram shows how the three tools, USE, EMF and ATL are used to calculate coverage measures for UML class diagrams. The measurable entities are captured within the ATL coverage calculation module.

As a trivial example to distinguish approaches I-III, consider the task of counting the number of classes in a UML class diagram. Following approach I we would simply define an OCL query on the class diagram metamodel to count and return the number of classes. Following approach II, we would define a transformation that would count the classes, and its target metamodel would be instantiated with the relevant name-value pair to represent the result. Finally, following approach III, we would first define a transformation to project the relevant classes to a metamodel of measurable entities, and the metrics themselves would be defined as queries over that metamodel.

We believe the third approach has a number of advantages. First, it clearly separates the three steps of the measurement process, which, following Kühne [13], we identify as a *projection* of the relevant details, an *abstraction* on the elements, and finally a *translation* to the numeric metric values. A second advantage is that it separates the process of metric specification from the application of the metrics tool, so that specification can be done independently, by mining the metric definitions and constructing an appropriate metamodel. A third advantage is pragmatic: by assembling the measurable entities as a separate model, comprehension and debugging of the measurement process is greatly facilitated.

In this paper we describe the use of ATL (ATLAS Transformation Language) in extending our previous work [14], based on approach III, to calculating coverage data from UML class and object diagrams. We apply the same approach of explicitly representing the measurable entities, and then calculating the coverage data from this. As well as the advantages listed above, this is of even greater importance for coverage measurement, since identifying which model entities have been covered would be considered a vital part of any coverage report.

2 Implementation

As a first step to implement a coverage tool for UML class diagrams we have implemented a tool-chain to calculate Generalisation (GN) coverage [3], for any

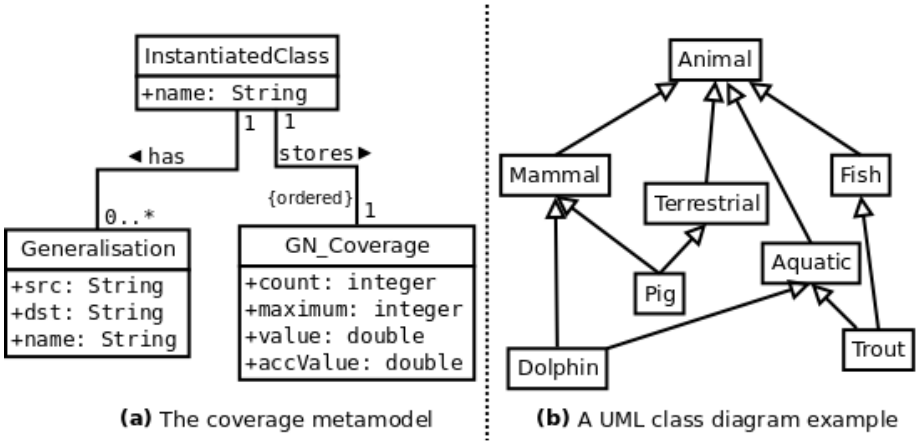

Fig. 2. UML class diagrams showing the coverage metamodel and an example input

class diagram and its corresponding object diagrams. The tool-chain ties together three tools:

USE (a UML-Based Specification Environment) is a tool that used for modelling and validating designs, and allows the user to create class and object diagrams (as well as other UML diagrams), and can also evaluate OCL constraints and expressions [15].

EMF (Eclipse Modeling Framework) is framework within the Eclipse IDE that provides code generation and tool support for modelling and metamodelling.

ATL is a model transformation language that transforms models based on OCL-like rules [16].

Figure 1 shows how USE, EMF and ATL are linked together in a tool-chain to perform coverage calculations. The inputs to the tool chain are a class diagram and one or more corresponding object diagrams, specified using the USE syntax. The output is an XML file representing an instance of our coverage metamodel.

Our tool-chain essentially uses the USE tool as a parser and validator for the class and object diagrams. This seemed more efficient than using other UML tools since USE is open-source, provides a good GUI to allow viewing of object and class diagrams, and provides many more facilities than shown here, which we hope to exploit in the future.

We have modified the USE tool in two ways. First we have annotated the USE Java source code that represents a UML metamodel, and used this as input to EMF to generate a UML metamodel in ECore format. Second we have implemented a visitor that walks an object diagram and generates the Java code that, when compiled and run, instantiates our EMF metamodel.

As shown in Figure 1, the output of the USE layer in our tool chain is a class diagram and a corresponding object diagram, both represented as instances of a corresponding ECore metamodel in EMF.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
  xmlns:org.nuim.lunar="http:///org/nuim/lunar.ecore">
  <instantiatedClass name="Dolphin">
    <generalisation src="Dolphin" dst="Aquatic" name="Dolphin"/>
    <generalisation src="Aquatic" dst="Animal" name="Dolphin"/>
    <generalisation src="Dolphin" dst="Mammal" name="Dolphin"/>
    <generalisation src="Mammal" dst="Animal" name="Dolphin"/>
    <GN_Coverage count="4" maximum="10" value="0.4" accValue="0.4"/>
  </instantiatedClass>
  <instantiatedClass name="Pig">
    <generalisation src="Pig" dst="Terrestrial" name="Pig"/>
    <generalisation src="Terrestrial" dst="Animal" name="Pig"/>
    <generalisation src="Pig" dst="Mammal" name="Pig"/>
    <generalisation src="Mammal" dst="Animal" name="Pig"/>
    <GN_Coverage count="4" maximum="10" value="0.4" accValue="0.8"/>
  </instantiatedClass>
  <instantiatedClass name="Trout">
    <generalisation src="Trout" dst="Aquatic" name="Trout"/>
    <generalisation src="Aquatic" dst="Animal" name="Trout"/>
    <generalisation src="Trout" dst="Fish" name="Trout"/>
    <generalisation src="Fish" dst="Animal" name="Trout"/>
    <GN_Coverage count="4" maximum="10" value="0.4" accValue="1.0"/>
  </instantiatedClass>
</xmi:XMI>
```

Fig. 3. An example of the XML output of our tool-chain, showing the *GN* coverage data for a simple object diagram corresponding to the class diagram in Figure 2(b).

2.1 Coverage Metamodel

In order to calculate the class model coverage data, a coverage metamodel shown in Figure 2(a) has been designed and used as the target model for an ATL transformation. The coverage metamodel contains a list of instantiated classes, each of which may have zero or multiple generalisation links. Each instantiated class stores relevant values in one $GN\_Coverage$. Then the ATL transformation takes in the necessary data from the class and object diagrams, *projects* out the countable entities, in this case generalisation links, *abstracts* to record instances of these from the object diagram, and uses ATL rules to *transform* this to the relevant coverage data.

Figure 2(b) shows a UML class diagram, taken from an introductory text on UML [17, pg 226]. This class diagram has 8 classes in total, with 10 generalisation links between these classes. Thus, by the generalisation criterion proposed by Andrews et al. [3], a test set ought to cover all 10 generalisation relationships.

In fact, to calculate $GN\_Coverage$ it is only necessary to consider whether or not a given class has been instantiated at least once. For example, knowing that the class Pig from Figure 2(b) has been instantiated allows us to conclude that the four generalisation links between Pig and $Animal$ have been covered. Figure 3 shows an example of the XML coverage file that is output for an object diagram containing at least one instantiation of $Dolphin$, Pig and $Trout$.

3 Future Work

At present our tool chain is limited in functionality as it only calculates generalisation coverage and a restricted version of association coverage. However, we believe that as a prototype it demonstrates the feasibility of the approach. We are currently working on an extension of the tool to handle other features of UML class and object diagrams, and hope eventually to harness the OCL constraints to assist in completing the coverage measures for class diagrams.

References

1. Pilskalnsa, O., Andrews, A., Knight, A., Ghosh, S., France, R.: Testing UML designs. Information and Software Technology **49**(8) (2007) 892–912
2. Utting, M., Legeard, B., eds.: Practical Model-Based Testing. Elsevier (2007)
3. Andrews, A., France, R., Ghosh, S., Craig, G.: Test adequacy criteria for UML design models. Soft. Test. Verif. & Reliability **13**(2) (2003) 95–127
4. McQuillan, J., Power, J.: A survey of UML-based coverage criteria for software testing. Technical Report NUIM-CS-TR-2005-08, NUI Maynooth (2005)
5. Dinh-Trong, T.T., Ghosh, S., France, R.B.: A systematic approach to generate inputs to test UML design models. In: 17th International Symposium on Software Reliability Engineering, Raleigh, NC (2006) 95–104
6. Mahdian, A., Andrews, A.A., Pilskalns, O.: Regression testing with UML software designs: a survey. J. of Software Maintenance and Evolution: Research and Practice **21**(4) (2009) 253–286
7. Briand, L., Labiche, Y., Lin, Q.: Improving the coverage criteria of UML state machines using data flow analysis. Soft. Test. Verif. & Reliability **20** (2010)
8. Baroni, A., Braz, S., Abreu, F.: Using OCL to formalize object-oriented design metrics definitions. In: ECOOP Workshop on Quantative Approaches in Object-Oriented Software Engineering, Malaga, Spain (2002)
9. Vépa, E., Bézivin, J., Brunelière, H., Jouault, F.: Measuring model repositories. In: Model Size Metrics Workshop at the MoDELS/UML, Genoa, Italy (2006)
10. OMG: Architecture-driven modernization (ADM): Software metrics meta-model (SMM). Beta Specification ptc/2009-03-03, Object Management Group (2009)
11. Mens, T., Lanza, M.: A graph-based metamodel for object-oriented software metrics. Electronic Notes in Theoretical Computer Science **72**(2) (2002) 57–68
12. Vépa, E.: ATL transformation example: UML2 to Measure. Available on-line as http://www.eclipse.org/m2m/atl/atlTransformations/#UML22Measure (2007)
13. Kühne, T.: Matters of (meta-)modeling. Software and System Modeling **5**(4) (2006) 369–385
14. McQuillan, J.A., Power, J.F.: A metamodel for the measurement of object-oriented systems: An analysis using Alloy. In: IEEE International Conference on Software Testing Verification and Validation, Lillehammer, Norway (2008) 288–297
15. Gogolla, M., Büttner, F., Richters, M.: USE: A UML-based specification environment for validating UML and OCL. Sci. Comp. Prog. **69**(1-3) (2007) 27–34
16. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. Sci. Comp. Prog. **72**(1-2) (2008) 31–39
17. Oestereich, B.: Developing Software with UML: Object-oriented analysis and design in practice. Addison-Wesley Professional (1997)