Towards Systematic Mutations for and with ATL Model Transformations*

Patrick Sommer[1] and Carola Gabriel[2] and Martin Keiblinger[3]

[1] Mautner Markhof-Gasse 58/4/31, 1110 Wien
e0925011@student.tuwien.ac.at
MatrNr.: 0925011
[2] Mustergasse 54/4/3, 1030 Wien
matthias@tuwien.ac.at
MatrNr.: 0426553
[3] Mustergasse 54/4/3, 1030 Wien
matthias@tuwien.ac.at
MatrNr.: 0426553

Abstract. This abstract summarizes the content of this paper in about 70 to 150 words. . . .

* This work has been created in the context of the course "188.952 Advanced Model Engineering (VU 2,0)" in SS15.

Table of Contents

1 Introduction

The broader goal of this work is to create an introduction to a specific field of software testing. Software testing is a process, or a series processes engineered to check if a program does what it is designed to do and that id does not do anything unintended. [10] Model based testing (MBT) is a variant of testing. Test cases are not written by the programmer directly. The programmer creates a model of the requirements and in a second step the test cases are generated on base of the model. [14]

Mutation testing is a fault-based testing technique. It applys changes to the input and creates a mutant. A mutant represents a faulty program. In the best case these changes, which are applied by the mutator, represent mistakes a programmer would make. [6]

The basic idea is of mutation testing is not to test the resulting software itself but the test cases. Good test cases should be able to identify mutants. Identifying means recognizing differing results of the original system under test (SUT) or mutants. [9]

The process of mutation testing consists of these components:

- *Test data* as input for the original programm P and its mutants.
- The original program P
- The *mutants* of P.
- An *oracle* which is able to decides if results differ and which is therefore able to identify mutants.

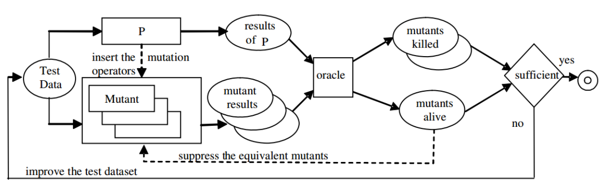

Fig. 1. The mutation testing workflow contains a feedback loop. [9]

The goal of the process1 is to *kill* or identify faulty versions of P. If a mutant outputs the same data as the P for the same input data it's called *equivalent*. In this case this mutant has to removed from the set of mutants under test.

The last step is to assess how good the tests are and check if they should be improved. Assume KM as the set of the killed mutants, M is the set of all mutants and EM is the set of all identified, equivalent mutants. Then the mutation score MS is calculated like this: [6]

$$MS = \frac{|KM|}{|M| - |EM|} \tag{1}$$

If this value is to small the tests have to be improved.

The success of this method depends on the set of mutants used in the process. Manual creation of mutants is a tedious and time consuming task. Therefore a quick, reliable and efficent creation of mutants is proposed in [5].

Troya et. al. builds upon ATL and higher order transformations (HOT) to create transformations to automatically generate mutants.

2 General

Model transformations play an important role in the Model Driven Engineering (MDE) approach. Developing model transformation definitions is expected to become a common task in model driven software development. [2] In this part of the paper we want to explain the basics of the requirements we needed for Mutations for and with ATL Model Transformations.

2.1 Model transformation

Model transformation is an important technique in software development, espacially in Model-Driven Software Development (MDSD) and Model-Driven Software Development (MDA). There exists different types of model transformations like Model-To-Model Transformation and Model-To-Text Transformation.

2.2 ATL

ATL is a model transformation language containing a mixture of declarative and imperative constructs. ATL is applied in the context of the transformation pattern shown in . In this pattern a source model Ma is transformed into a target model Mb according to a transformation definition mma2mmb.atl written in the ATL language. The transformation definition is a model conforming to the ATL metamodel. All metamodels conform to the MOF. ATL is a hybrid transformation language. It contains a mixture of declarative and imperative constructs. We encourage a declarative style of specifying transformations. The declarative style of transformation specification has a number of advantages. It is usually based on specifying relations between source and target patterns and thus tends to be closer to the way the developers intuitively perceive a transformation. This style stresses on encoding these relations and hides the details related to selection of source elements, rule triggering and ordering, dealing with traceability, etc. Therefore, it can hide complex transformation algorithms behind a simple syntax. [2]

[2]

[2]

[2]

In the following you can see a short example of a ATL transformation:

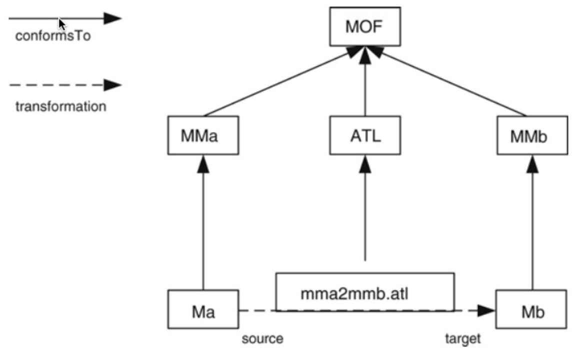

Fig. 2. Overview of the ATL transformational approach

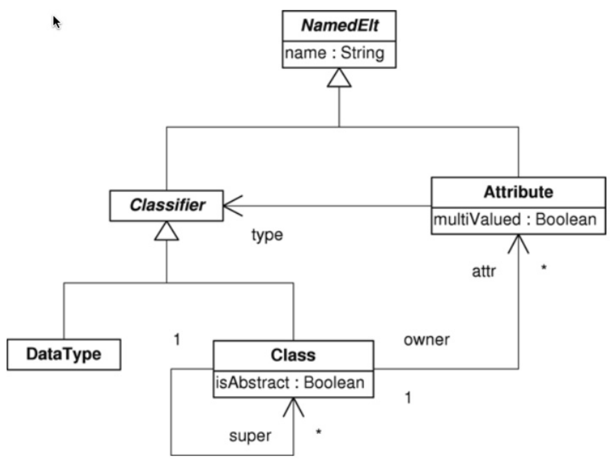

Fig. 3. Class metamodel

```
//Start Program
module Entities2Forms;
create OUT : Forms from IN : Forms;

rule EntityModel2FormModel {
from
em : Forms!EntityModel
to
fm : Forms!FormModel (
)
}
//End Program
```

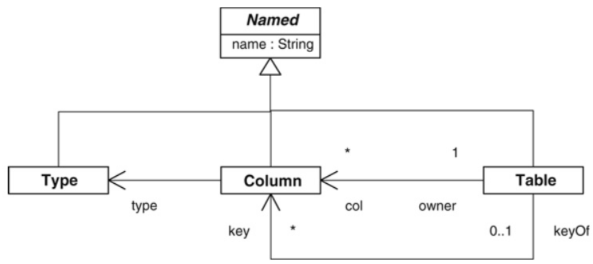

Fig. 4. Relational metamodel

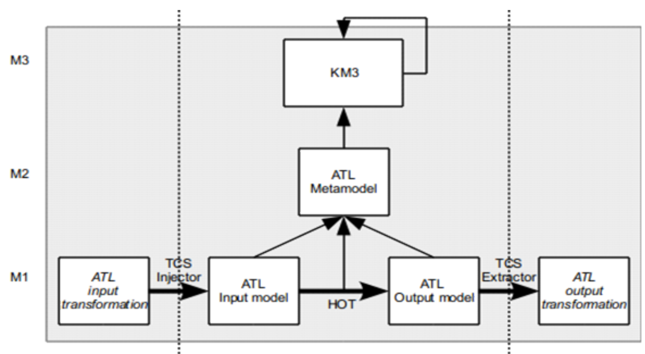

Fig. 5. Sample schema of a HOT for transformation modification in ATL

This ATL file shows a tranformation Entities2Form. The source model is the 'IN' model and the target model is 'OUT'. The rule mapped the elements EntityModel to FormModel.

2.3 High Order Transformations

High Order Transformations are defined in some special languages of model transformations. That means that the model transformation is itself a model. The input and output models are also themselves transformation models.

[7]

3 Implementation

For working with LaTeX you can take advantage of a variety of books and free introductions and tutorials on the internet. A competent contact point for LaTeX beginners is the LaTeX Wikibook, which is available under `http://en.wikibooks.org/wiki/LaTeX`.

The following sections give examples of the most important LaTeX environments and commands.

3.1 Tables

Tables have to be realized with the help of the *table* environment. Tables shall be sequentially numbered for each chapter and described in terms of a short caption (cf. Table 1).

Name	**Date**	**Title**
Mustermann Adam	18.5	T1
Musterfrau Eva	22.6	T2

Table 1. Seminar for Master Students

3.2 Figures

Like tables, figures shall be sequentially numbered for each chapter and described in terms of a short caption). You could either produce your drawings directly inside Latex using PSTricks[4], Tikz[5], or any set of macros dedicated to your requirements (cf. Figure 6). Alternatively, you may include figures prepared in external tools (cf. Figure 7). Note, to ensure high quality printing, all figures must have at least 300 dpi.

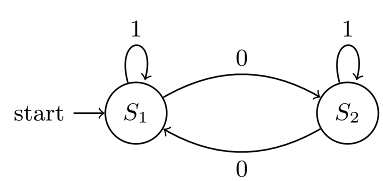

Fig. 6. Sample figure

[4] `http://tug.org/PSTricks`
[5] `http://sourceforge.net/projects/pgf`

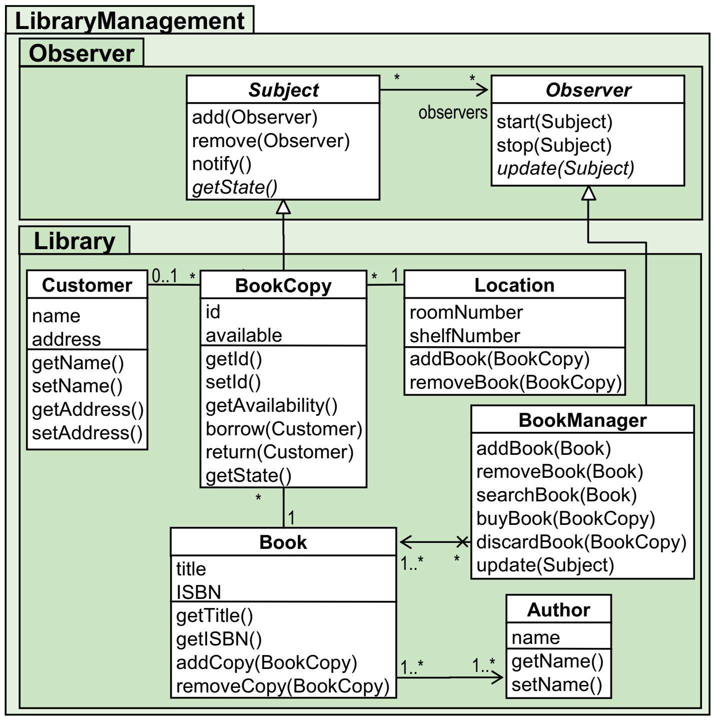

Fig. 7. Sample figure

3.3 Fonts

When introducing important terms for the first time use *emphasize*. For a consistent look and feel of proper names like Class Diagram and `Observer` pattern you may define macros in the main document `thesis.tex`.

3.4 Code

For short code fragments use the *verbatim* environment.

```
//Start Program
System.out.println("Hello World!");
//End Program
```

A much better alternative is the *algorithm* environment (cf. Algorithm 1.1). This environment offers special formatting features for loops, operations and comments.

```
input  : A bitmap Im of size w × l
output: A partition of the bitmap
special treatment of the first line;
for i ← 2 to l do
    special treatment of the first element of line i;
    for j ← 2 to w do
        left ← FindCompress(Im[i, j − 1]);
        up ← FindCompress(Im[i − 1, ]);
        this ← FindCompress(Im[i, j]);
        if left compatible with this then ;                    // O(left,this)==1

            if left < this then Union(left,this);
            ;
            else Union(this,left);
            ;
        end
        if up compatible with this then ;                      // O(up,this)==1

            if up < this then Union(up,this);
            ;
            // this is put under up to keep tree as flat as possible
            else Union(this,up);
            ;                                                  // this linked to up
        end
    end
    foreach element e of the line i do FindCompress(p);
end
```

Algorithm 1.1: Sample algorithm

4 Bibliographic Issues

4.1 Literature Search

Information on online libraries and literature search, e.g., interesting magazines, journals, conferences, and organizations may be found at `http://www.big.tuwien.ac.at/teaching/info.html`.

4.2 BibTeX

BibTeX should be used for referencing.

The LaTeX source document of this pdf document provides you with different samples for references to journals [4], conference papers [12], books [3], book chapters [13], electronic standards [11], dissertations [15], masters' theses [8], and web sites [1]. The respective BibTeX entries may be found in the file `references.bib`. For administration of the BibTeX references we recommend `http://www.citeulike.org` or JabRef for offline administration, respectively.

References

1. Business Informatics Group. http://www.big.tuwien.ac.at. Accessed: 2010-11-09.
2. Frédéric Jouault, Freddy Allilaire, J. B., and Kurtev, I. Atl: A model transformation tool. *Software Engineering Group 37*, 5 (Aug 2007), 31–39.
3. Hitz, M., Kappel, G., Kapsammer, E., and Retschitzegger, W. *UML @ Work, Objektorientierte Modellierung mit UML 2*, 3. ed. dpunkt.verlag, 2005 (in German).
4. Huemer, C., Liegl, P., Schuster, R., and Zapletal, M. B2B Services: Worksheet-Driven Development of Modeling Artifacts and Code. *Computer Journal 52*, 2 (2009), 28–67.
5. Jia, Y., and Harman, M. An analysis and survey of the development of mutation testing. *Software Engineering, IEEE Transactions on 37*, 5 (Sept 2011), 649–678.
6. Jordi Cabot. http://modeling-languages.com/introducing-higher-order-transformations-hots/. Accessed: 2015-04-15.
7. Langer, P. Konflikterkennung in der Modellversionierung. Master's thesis, Vienna University of Technology, 2009.
8. Mottu, J.-M., Baudry, B., and Le Traon, Y. Mutation analysis testing for model transformations. In *Proceedings of the Second European Conference on Model Driven Architecture: Foundations and Applications* (Berlin, Heidelberg, 2006), ECMDA-FA'06, Springer-Verlag, pp. 376–390.
9. Myers, G. J., and Sandler, C. *The Art of Software Testing.* John Wiley & Sons, 2004.
10. OASIS. *Business Process Execution Language 2.0 (WS-BPEL 2.0)*, 2007.
11. Schauerhuber, A., Wimmer, M., Schwinger, W., Kapsammer, E., and Retschitzegger, W. Aspect-Oriented Modeling of Ubiquitous Web Applications: The aspectWebML Approach. In *Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS '07), March 26-29, Tucson, Arizona, USA* (2007), IEEE CS Press, pp. 569–576.
12. Schwinger, W., and Koch, N. Modeling Web Applications. In *Web Engineering*, G. Kappel, B. Pröll, S. Reich, and W. Retschitzegger, Eds. John Wiley & Sons, Ltd, 2006, pp. 39–64.
13. Troya, J., Bergmayr, A., Burgueo, L., and Wimmer, M. Towards systematic mutations for and with atl. In *Proc. of the Eighth IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW); 10th International Workshop on Mutation Analysis (Mutation 2015)* (2015).
14. Utting, M., Pretschner, A., and Legeard, B. A taxonomy of model-based testing approaches. *Softw. Test. Verif. Reliab. 22*, 5 (Aug. 2012), 297–312.
15. Wimmer, M. *From Mining to Mapping and Roundtrip Transformations - A Systematic Approach to Model-based Tool Integration.* PhD thesis, Vienna University of Technology, 2008.