ATL Demystified and an Introduction to the RPG2Petrinet Experiment

Daan Janssens

daan.janssens@student.uantwerpen.be

University of Antwerp

Abstract

ATL is a fairly well known M2M model transformation language. It's a hybrid rule-based domain-specific language (mixes declarative & imperative constructs) and is part of the Eclipse Modeling Project. In this reading report we will introduce the ATL language and take a look at various special aspects and constructs introduced in multiple other ATL related papers. We'll try to partially demystify the broad spectrum of possibilities offered by ATL!

Keywords: ATL, eclipse, M2M, in-place, traceability, RPG, Petrinet

1. Introduction

Model transformations play the key role in Model Driven Engineering. It seems ATL(ATLAS Transformation Language) fits perfectly in this picture. In this report we'll try to show why this is the case. It bundles multiple other ATL related papers together and offers a short but useful introduction to the language. This report is organized as follows. Section 2 gives a short introduction to the basics of ATL (mainly based on Jouault et al. (2008)). Section 3 covers traceability for ATL (based on Jouault (2005)). Section 4 covers In-Place transformations (based on Tisi et al. (2011)) and explains why we didn't chose an experiment like modeling the operational semantics of an RPG game. Section 5 presents our first experience with the ATL eclipse modeling environment (based on Allilaire et al. (2006)). Lastly, section 6 covers a short introduction to our planned experiment.

2. General overview of ATL

ATL follows a specific transformation pattern: a (read-only) source model Ma gets transformed to a (write-only) target model Mb. This happens according to a transformation definition MMa2MMb, which is written in ATL. This transformation definition itself can be seen as a model which conforms to the ATL metamodel, while Ma and Mb conform respectively to the metamodels MMa and MMb. All metamodels conform to the MOF metamodel.

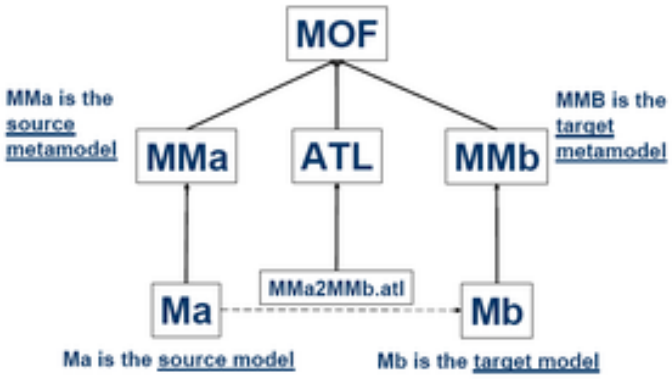

Figure 1: The transformation pattern

In Jouault et al. (2008) the basic syntax of ATL is explained. It covers helpers, which are basically functions with expressions written in OCL. They can navigate across model elements, though only on the read-only source model side.

A transformation definition bundles one or multiple rules. Those rules are the heart of ATL transformations(Allilaire et al. (2006)) because they describe how output elements are produced from an input element. They can be specified in a declarative(matching rules) or imperative way. The declarative approach is most of the time encouraged Jouault et al. (2008).

A matched rule is composed of a source pattern and a target pattern. The source pattern(also called inPattern (Bzivin et al. (2003))) specifies a set of source types (from the source metamodel) and possibly a boolean OCL expression (guard). The ATL engine will try to find a set of matches of this pattern in the source model. The target pattern(outPattern) is composed of a set of elements, where every element specifies a target type(from the

target metamodel) and a set of bindings. When a rule containing the target pattern is executed, the target elements of the specified types are created. The binding will specify the value used to initialize the properties.

There can be an entrypoint rule, which is the rule being executed first, this rule can call other rules (imperative way). Also rule inheritance exists (the new rule can specify additional elements or restrictions), however according to Kusel et al. (2013) the support of it in ATL is still limited.

Most papers offered one or multiple case-studies, which have been a great help in understanding the syntax. (Jouault et al. (2008), Bzivin et al. (2003), Jouault (2005))

3. Traceability in ATL

By executing rules on a match, a dynamic traceability link will be automatically created in the internal strutures of the transformation engine(Jouault et al. (2008)). This link relates three components: the rule, the match and the newly created target elements.

However if we want a persistent version of the traceability links we have to find another approach. This approach is explaind in Jouault (2005). They do this by attaching traceability generation code to pre-existing ATL programs, this is possible because we can consider the transformation definition, as well as the traceability information as a model. In this way the traceability generation code can be attached after a program is written.

Traceability is not in the scope of our planned experiment, but reading this paper gave us an idea of the power of ATL. They also showed examples and introduced HOT's (Higher-Order Transformations).

4. In-Place transformations in ATL

We initially planned implementing the operational semantics of an RPG(Role-Playing Game) game, based on a RPG metamodel and model, as ATL experiment. New versions of ATL do support In-Place transformations, so that's not really the problem. ATL however doesn't seem to support a step-wise execution. It will try to find all matches for all rules and apply these, this

means you'll go from initial state to final state in just a blink. This is not the case in some other tools, like AtomPM(Syriani (2009)).

ATL was primary designed to transform read-only input models towards write-only output models. This is not always exactly what we want. In some cases we want to modify the source model (e.g. refactorings). This is called In-Place transformations. The work of Tisi et al. (2011) described in great detail the process of implementing In-Place transformations.In that paper they refer to it by the term 'refinement transformations' or 'refinement mode'

Before ATL supported refinement mode this had to be simulated by copying all source elements to the target elements and making the small changes while copying. This naive solution had a lot of drawbacks (e.g execution time increases.) In Tisi et al. (2011) they describe an alternative way that overcomes these limitations, namely the in-place transformation mode. Where the idea of constructing the target model incrementally from an empty one is abandoned and the transformation starts from the whole source model.

5. First experience with the ATL eclipse modeling environment

The ATL eclipse modeling environment has been working like a charm so far, however installing it and getting acquainted is a different story. There is a lot of outdated information on the internet, which can be misleading. The paper of Allilaire et al. (2006) didn't really help a lot on this end. Also, Models have to be made in xmi, which lacks the graphical aspect, which makes the job of making the model cumbersome and increases the chances of mistakes.

We ended up getting acquainted with the plugin from a series of youtube videos (abidredlove (2009)) that explained the basics of the environment and the installation of it.

6. The RPG2Petrinet experiment

As explained in section 4, we chose not to implement the operational semantics of an RPG game, but focus on the denotational part. Namely, transforming an RPG metamodel to a petrinet. This has been briefly described in the work of Marques et al. (2012). We will use the same approach:

first transform to a intermediate metamodel then transform to the petrinet metamodel.

This planned experiment will cover a basic implementation of a transformation from a modeled role-playing game into a matching petrinet model which can be used for analyzing purposes lateron. We will start with the RPG Language Meta-Model which describes every possible feature of the defined RPG Domain. After that we will build another intermediate language (μRPG Language) which will work as a filter. It will only contain the essential entities needed to check properties related to the petrinet analysis. Lastly, a transformation for RPG2μRPG and μRPG2petrinet will be implemented!

References

abidredlove, 2009. abidredlove youtube channel @ONLINE. URL: http://www.youtube.com/user/abidredlove?feature=watch.

Allilaire, F., Bzivin, J., Jouault, F., Kurtev, I., 2006. Atl – eclipse support for model transformation, in: IN: PROC. OF THE ECLIPSE TECHNOLOGY EXCHANGE WORKSHOP (ETX) AT ECOOP.

Bzivin, J., Dup, G., Jouault, F., Pitette, G., Rougui, J.E., 2003. First experiments with the atl model transformation language: Transforming xslt into xquery, in: 2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture.

Jouault, F., 2005. Loosely coupled traceability for atl, in: In Proceedings of the European Conference on Model Driven Architecture (ECMDA) workshop on traceability, pp. 29–37.

Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., 2008. Atl: A model transformation tool. Sci. Comput. Program. 72, 31–39. URL: http://dx.doi.org/10.1016/j.scico.2007.08.002, doi:10.1016/j.scico.2007.08.002.

Kusel, A., Schönböck, J., Wimmer, M., Retschitzegger, W., Schwinger, W., Kappel, G., 2013. Reality check for model transformation reuse: The atl transformation zoo case study, in: 2nd Workshop on the Analysis of Model Transformations (AMT) @ MODELS'13, 2nd Workshop on the Analysis of Model Transformations (AMT). pp. 1–10.

Marques, E., Balegas, V., Barroca, B.F., Barisic, A., Amaral, V., 2012. The rpg dsl: A case study of language engineering using mdd for generating rpg games for mobile phones, in: Proceedings of the 2012 Workshop on Domain-specific Modeling, ACM, New York, NY, USA. pp. 13–18. URL: http://doi.acm.org/10.1145/2420918.2420923, doi:10.1145/2420918.2420923.

Syriani, E., 2009. Atompm @ONLINE. URL: http://syriani.cs.ua.edu/atompm/atompm.htm.

Tisi, M., Martínez, S., Jouault, F., Cabot, J., 2011. Refining Models with Rule-based Model Transformations. Rapport de recherche RR-7582. INRIA. URL: http://hal.inria.fr/inria-00580033.