A comparison of AToMPM and GROOVE

Brent van Bladel

University of Antwerp, Belgium

Abstract

Domain-specific visual modelling is a relatively new area in the software engineering. The development of this methodology is accompanied by a need for good visual modelling tools. This paper compares the capabilities and features of two modelling tools: AToMPM and GROOVE.

Keywords:
visual modelling language, graph transformation, Groove, AToMPM

1. Introduction

Since object-oriented programming was introduced in the 1960s, there haven't been many changes to the way we program. Recently, however, the concept of using domain-specific (visual) modelling to generate code is becoming more popular. As this popularity grows, the need for good visual modelling tools grows as well. This paper describes the differences between two modelling tools: AToMPM and GROOVE.

AToMPM stands for "A Tool for Multi-Paradigm Modeling". It is an open-source framework for designing DSML environments, performing model transformations, and manipulating and managing models [1].

GROOVE stands for "GRaph-based Object-Oriented VErification". It is centered around the use of simple graphs for modelling object-oriented systems, and graph transformations as a basis for model transformation and operational semantics [2].

This paper will compare these two tools. Section 2 shows how both tools can be used for modelling and metamodelling. Section 3 handles the operational semantics using transformation rules. The work I plan to do for this project is described in section 4. We conclude with the different advantages offered by each tool in section 5.

2. Models and Graphs

2.1. Metamodelling

AToMPM allows the user to define their own metamodel in the Class Diagram formalism. Such a metamodel consist of a set of classes defining the objects that can be used in the model. Each class can also contain attributes, constraints and actions (in the javascript programming language). Classes can be connected with each other signifying relations between them. It is also possible to define global constraints in javascript.

GROOVE's counterpart of metamodelling is a type graph. The type graph specifies the allowed structure of the graphs, as well as the node type hierarchy, similar to a class diagram. It is not required to use a type graph in GROOVE. In untyped mode the host graphs are not constrained at all.

2.2. Modelling

A model in AToMPM can conform to any metamodel you define. It is even necessary to design a concrete syntax for your metamodel. Therefore there is almost no limit on what you are able to model in AToMPM. The models in GROOVE are more constrained. They must be graphs consisting of labelled nodes and edges. In typed mode, all graphs conform to the type graph which determines the allowed combinations of node types and edges [3].

Objects can have attributes, which are visually represented in Groove. The attribute values are each represented by a single node. The label on the edge connecting that node to the object is the attribute name. [4] Attributes in AToMPM are purely textual. They can be set when editing the properties of an object using a combination of the python and javascript programming languages.

3. Model transformation

3.1. Transformation rules

In order to implement operational semantics, the tool needs to provide functionality to transform the models in specified patterns.

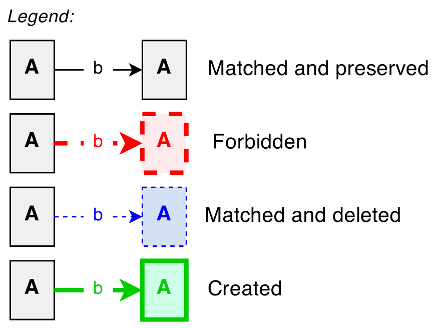

Figure 1: Groove transformation rule legend

In GROOVE, graphs are transformed by applying rules. There are four types of nodes and edges in such rules, distinguished by their colours and shapes (see figure 1):

- A pattern that must be present in the host graph in order for the rule to be applicable. This pattern is represented as black (continuous thin) nodes and edges. They must be present and are preserved.

- Subpatterns that must be absent in the host graph in order for the rule to be applicable. This pattern is represented as red (dashed fat) nodes and edges.

- Elements (nodes and edges) to be deleted from the graph. They are represented as blue (dashed thin) nodes and edges. These elements must be present and are deleted;

- Elements (nodes and edges) to be added to the graph. They are represented as green (continuous fat) nodes and edges. These elements are created.

All these elements are combined into a single graph shows a small example containing most of these elements [3; 5].

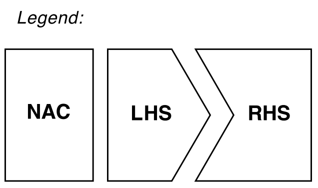

Figure 2: Atompm transformation rule legend

Transformation rules in AToMPM are composed of one Right-Hand Side (RHS) pattern, one Left-Hand Side (LHS) pattern and zero or more Negative Application Condition (NAC) patterns (see figure 2) [6]. The pattern in the LHS will be matched in the host model and replaced by the pattern in the RHS. The NAC contains patterns that must be absent in the host model in order for the rule to be applicable.

As you can see, both tools provide the same concept of transformation rules in two completely different ways. They are, however, equivalent and the same rule can be modelled in each formalism without loss of meaning. This can be shown easily with an example (figure 3).

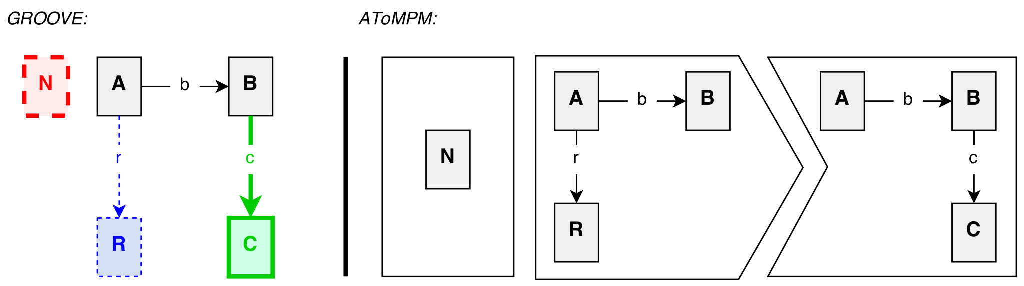

Figure 3: Comparison of a transformation rule in GROOVE and AToMPM

The transformation rule in this example tries to match the pattern formed by objects A and B. We recognize this as the black (continuous thin) nodes and edges in GROOVE. This pattern appears in both the LHS and the RHS in the AToMPM representation of the rule. Therefore it is matched and preserved. Object R is removed, which is shown by the blue (dashed thin) node in GROOVE. This is represented in AToMPM by the fact that object

R appears in the LHS, but is absent in the RHS. When creating an object this is reversed. Object C appears in the RHS, but is absent in the LHS, and therefore it is created. This is shown in GROOVE with the green (continuous fat) node. Finally, the example states that object N must be absent in order for the rule to be applicable. GROOVE represents this as a red (dashed fat) node, while AToMPM provides the NAC for this functionality.

3.2. Attribute manipulation

Because GROOVE implements attributes as part of the graph, operations on these attributes must also be implemented as part of that graph. One node represents the operation, two nodes represent the operands on which the operation must be applied and one node represents the result [4]. Figure 4 shows an example of how an attribute can be incremented in GROOVE.

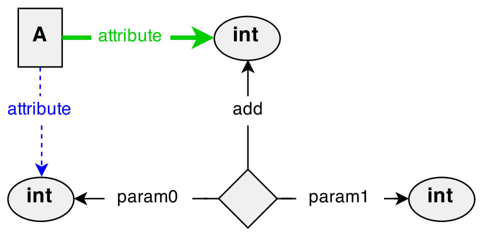

Figure 4: Example of an operation in GROOVE

Operations on attributes in AToMPM must be implemented in the python programming language. For example, to increment an attribute in AToMPM, python code that calculates the addition should be written in the RHS of the transformation rule. The attribute will be set to the result of the calculation because the object appears in the RHS and thus in the pattern that will replace the LHS.

3.3. Control language

When it comes to operational semantics, we generally want to apply more than one rule to our model. Normally, a set of rules is executed in a particular order and with some sort of control structure. Such a set of rules forms an algorithm that can simulate the model.

The standard behaviour of GROOVE is to attempt the application of arbitrary rules at any point in time. There are, however, two further methods to control and direct the application of rules. The more straightforward mechanism is to assign priorities to rules. Low-priority rules may only be applied if no higher-priority rule is applicable. A more sophisticated mechanism is to use GROOVEs control language. A control program is imposed on top of a graph transformation system and specifies the allowed order of application of the rules of that system [3].

GROOVE's control language is purely textual and resembles a standard programming language. AToMPM on the other hand provides a visual control language called MoTif [6]. A MoTif control program consists of rule elements which have one input and two outputs (figure 5). One of the outputs expresses successful application of the rule, the other expresses failure of the rule. The output of one such element can be connected to the input of another element. This determines the order in which the rules are applied. There is a variety of different rule elements allowing for more complex structures.

Figure 5: Example of a motif rule in AToMPM

In general, AToMPM uses more textual language constructs than GROOVE. The overall tendency to incorporate python and javascript code in the modelling environment makes AToMPM more powerful. It is therefore interesting to note that the control language is the only part of AToMPM which tends more towards visual modelling than its GROOVE counterpart. Also note that the control language is the only part were GROOVE uses textual constructs.

4. Future work

In order to perform further comparison of the two tools, I will perform a case-study in which I implement the same requirements in both in GROOVE and AToMPM. I will use the RPGame formalism, which implements a simple role playing game where a hero tries to reach a goal while avoiding enemies. This case study provides the opportunity to compare how the tools perform when modelling the same formalism.

5. Conclusion

It is clear that AToMPM and GROOVE are two quite different tools. To conclude we give an overview of the extra features offered by each tool.

Unlike AToMPM, GROOVE has the ability to perform analysis on its models. More specifically, it can perform state space exploration. The simulator will attempt to generate the full state space of a given model. This entails recursively computing and applying all enabled graph transformation rules at each state [5].

On the other hand, the fact that concrete syntax must also be modelled in AToMPM makes the tool more flexible. The models can visually represent the problem domain. The fixed visual representation of the graphs in GROOVE cause non-distinguishability between formalisms at first glance. Therefore one of the advantages of visual modelling is lost in GROOVE [7].

AToMPM counts on the python en javascript programming languages for different parts of its models. This makes AToMPM more powerful than GROOVE. However, it causes AToMPM to deviate from the core concepts of visual modelling. Finally we must remark that the close link between AToMPM and textual languages results in easy code generation from its models, which is not supported by GROOVE.

References

[1] E. Syriani, H. Vangheluwe, Atompm website, project description, http://www-ens.iro.umontreal.ca/~syriani/atompm/atompm.htm (2014).

[2] M. de Mol, A. Rensink, E. Zambon, Groove website, about, http://groove.cs.utwente.nl/about/ (2014).

[3] A. H. Ghamarian, M. de Mol, A. Rensink, E. Zambon, M. Zimakova, Modelling and analysis using groove, International Journal on Software Tools for Technology Transfer (STTT).
URL http://doc.utwente.nl/77423/

[4] H. Kastenberg, Towards attributed graphs in groove, proceedings of Workshop on Graph Transformation for Verification and Concurrency, GT-VC 2005 (2005).
URL http://doc.utwente.nl/54440/

[5] A. Rensink, The groove simulator: Atool for state space generation, in: J. Pfaltz, M. Nagl, B. Bhlen (Eds.), Applications of Graph Transformations with Industrial Relevance, Vol. 3062 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2004, pp. 479–485. doi:10.1007/978-3-540-25959-6_40.
URL http://dx.doi.org/10.1007/978-3-540-25959-6_40

[6] R. Mannadiar, Atompm users manual, atompm website (2012).

[7] D. Moody, The physics of notations: Toward a scientific basis for constructing visual notations in software engineering, Software Engineering, IEEE Transactions on 35 (6) (2009) 756–779. doi:10.1109/TSE.2009.67.