Transformation Rules with MetaEdit+

Travassos, Willer[1]

McGill University, Montreal QC H3A 1A1, Canada,
willer.travassos@mail.mcgill.ca,
WWW home page: http://www.cs.mcgill.ca

Abstract. The goal of this project is to implement tranformation rules from AToM3 to MetaEdit+. This paper starts with a short description of the differences in rule handling between AToM3 and MetaEdit+. This short introduction is followed by a discussion about implementing a meta-model, transformation rules for the simulation of Finite State Machines. The second part of this paper consists of the process of adapting the AToM3 meta-model, ruels and FSM simulation into the MetaEdit+'s framwork.

1 Introduction

Differently from AToM3, MetaEdit+ does not provide means to create and execute transformation rules using Graph Grammars. Transformation rules are powerful methods for model manipulation that opens new opportunities with Domain Specific Modeling (DSM) applications. Such an opportunity is, for example, an application that utilizes two different meta-models, which communicate with each other, to execute functions within this application.

Even though MetaEdit+ is lacks rule handling, it does provide an API that can be used to manipulate models, in a similar fashion to AToM3. The goal of this project is to adapt rules created in AToM3 to MetaEdit+. These rules would follow the logic of AToM3 rules and would also emulate the structures used by AToM3 to handle its models.

This project is split into two parts: first is related to work done with AToM3, and the second is related to work with MetaEdit+. The meta-model, and models generated from it, are based on Finite State Machine (FSM), while the test transformation rules, used on both applications, are related to the simulation of a FSM.

The work with AToM3 is divided into three parts: Meta-modeling a FSM, followed by creating rules for its simulation, and finally running and testing the FSM simulation. The second part of this experiment consists of meta-modeling the FSM in MetaEdit+, followed by testing API methods with a test FSM model, and adapting and applying AToM3 rules to MetaEdit+.

2 AToM3: Meta-Modeling Finite State Machines

The meta-modeling of Finite State Machines in AToM3 was based around the work done on the publication Meta-Models are models too, by Hans Vangheluwe and Juan de Lara. Differently from their publication, the FSM meta-model was developed using the Class Diagram meta-model provided in AToM3.

The FSM meta-model contained a class to represent States in a FSM, and an association class that represented Transitions connecting FSM states. Also, there is an additional class, called Current, and an association class, called Points_to, to help in the future simulation of a FSM. Once the FSM meta-model was finalized, a test model was created that was used as a testing environment for the transformation rules.

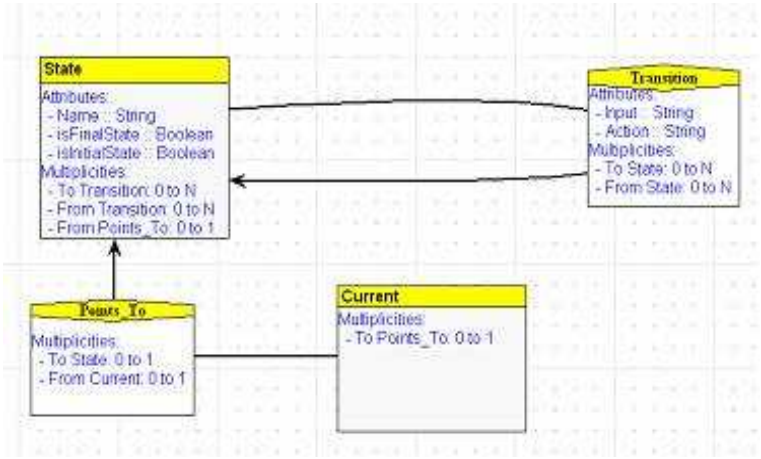

Fig. 1. Finite State Machine meta-model(Class Diagram)

3 AToM3: Implementing the FSM transformation rules

With the implementation of the FSM complete, transformations rules for this meta-model could be finally created. These rules are the ones that control the execution of the FSM simulation flow. They were based on the rules for FSM simulation described by the publication Meta-Models are models too.

There are three rules for simulation, and they correspond to starting the simulation, and carrying it on. It is here that the Current class and the Points_to association class from the FSM meta-model encounter their purpose. The idea of having these extra classes is to facilitate the tracking of the current state during the simulation of a FSM.

The simulation itself was to be carried by having AToM3 asking a user to input the next transition to be used. Once a transition was chosen, the current

state in the simulation would change to the state following that transition. The rules used are described as follows:

- LocateInitial: like its name says, it locates the initial state of any FSM model, and assigns the current state to it. It has the highest execution order to avoid trapping of the simulation run on its first state.
- StateTransition: this rule tells AToM3 how our simulation should behave when the current state of a FSM changed. It has the lowest execution.
- LocalStateTransition: worked in the same way to StateTransition but it was related to transition loops

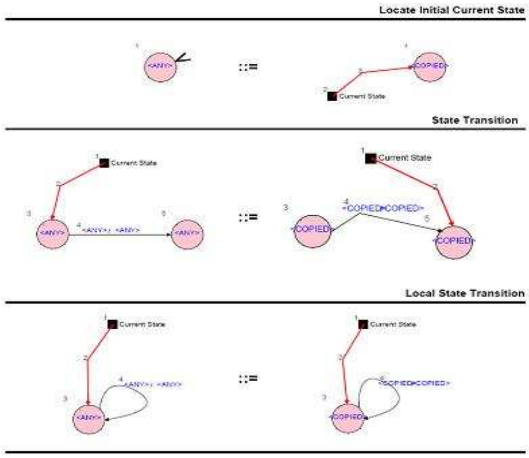

Fig. 2. Transformation Rules used for FSM simulation

The next step in this project was to finally simulate a FSM using the rules described above. Differently from the Meta-Models are models too publication, user input was not used in the simulation steps which made the simulation move randomly from one state to another. Also, since there was not a transformation rule that dealt simulation termination, which caused the simulation to restart, once there was no matching for these rules.

4 Working with MetaEdit+

Once the AToM3 side was finished, the next step in this project was to move this work into MetaEdit+. Thus, following the same process from the project's first part, an implementation of a FSM meta-model was necessary.

Using MetaEdit+'s modeling tools the parts of the FSM meta-model were created. Using the MetaEdit+'s graph tool the created meta-model parts were joined to create the actual FSM meta-model. With the meta-model set, a test model was created to be used in the API testing, and FSM simulation.

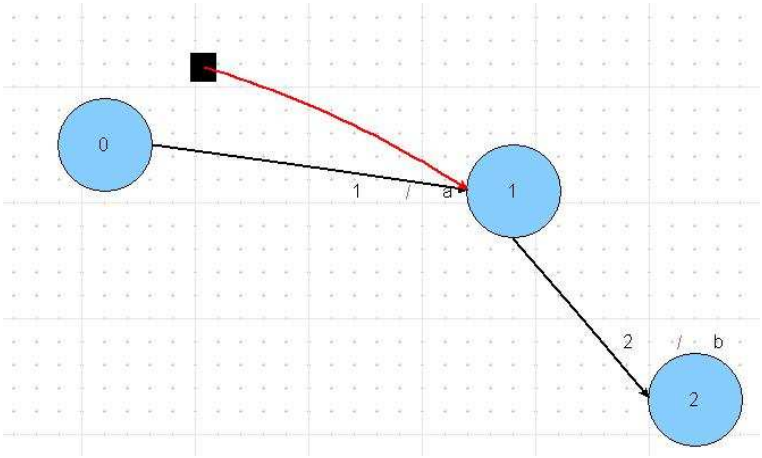

Fig. 3. A simulation step

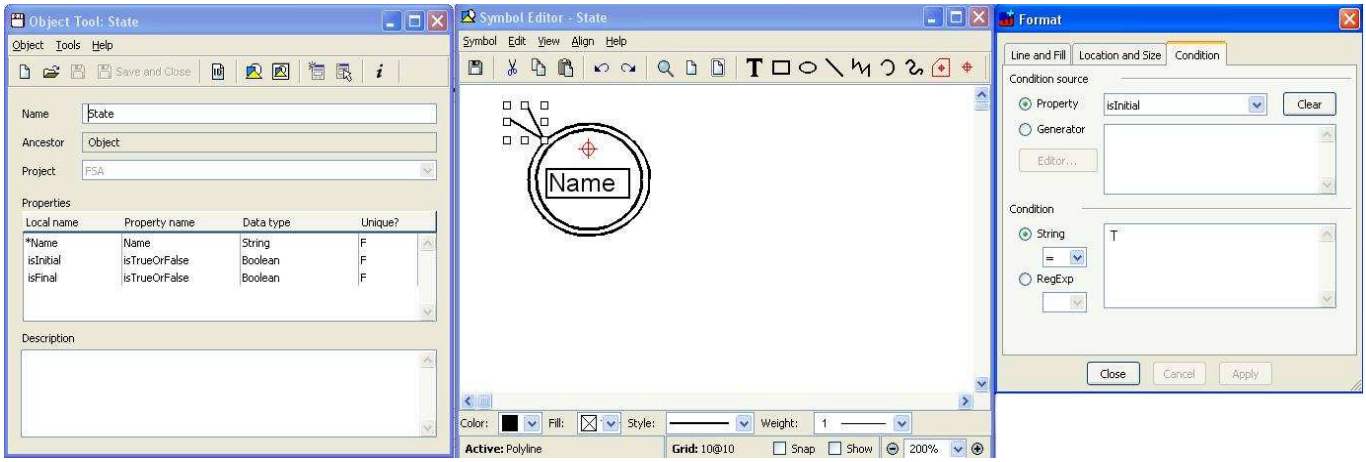

Fig. 4. A state class/object is being created. Dynamic display is being used on it, in the other to differentiate initial states (with top arrow), from the final states (bottom arrow).

The API testing was done using the model depicted in the figure below. These tests explored the capabilities of API methods to manipulate and extract data from models developed with MetaEdit+. They concentrated on three aspects of the MetaEdit+ API: animation of model, retrieval of models and its states and transition, and retrieval of model, state, and transition properties.

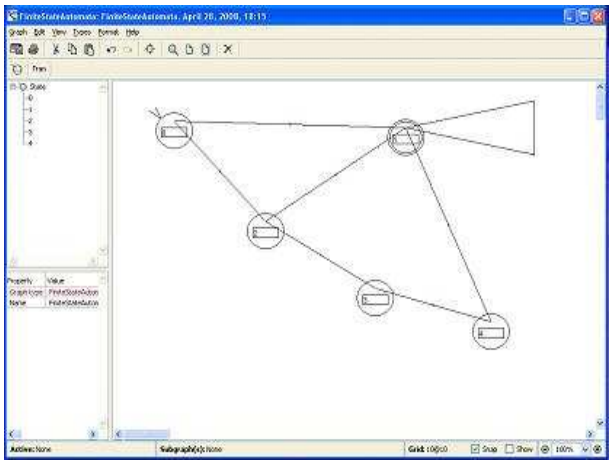

Fig. 5. Test Model

5 Adapting AToM3 rules into MetaEdit+

In order to start adapting the AToM3 rules into MetaEdit+, the author had to find methods to understand the work behind them. To makes this process simpler, a code generation tool called MOTIF, coded by Eugene Syriani, was used to generate python code that handle the Graph Grammars (GG) behind these rules. These generated python files were then used as the basis for the MetaEdit+ rules.

It is important to note here that the LocalStateTransition rule did not generate any python code that handled the rule. Also, this rule is very similar to the StateTransition rule, and it seemed natural to merge them together into one MetaEdit+ rule.

The MetaEdit+ API consists of web services that are defined in Web Services Definition Language file, provided by the API tool of MetaEdit+. This same file consists of data types and data structures specific to MetaEdit+, and all work with the API centers around them. For simplicity, the MetaEdit+ rules were also written in python code.

The adaptation of these rules did not go without problems though. As it was said, all the work that is done with API is centered on MetaEdit+'s own types.

These types did not provide access to the objects present in a model; instead, they provided references to them.

Thus, Instead of retrieving an object and all its properties, the author had to make a call to retrieve the object reference, and then use this reference to make a call to retrieve a property. If there was need to retrieve another property from the same object, another call would have to be made.

This excessive amount of calls helped increase the size of the program, and considerably slowed down the execution of the FSM simulation. Also, differently from AToM3 there is no access to the whole graph of a model. You can only retrieve parts of it, for example in the FSM meta-model, states and transitions were retrieved separately only. Then methods had to be used to find out how they were related to each other.

Another difficulty about adapting AToM3 rules was that the python SOAP libraries, used for projects presented several problems to the execution of this project. The first library, called SOAPpy, was unable to parse the API's WSDL file for the MetaEdit+ data types, which made the API use impossible.

The second library, called Zolera SOAP Infrastructure, did parse the WSDL file correctly but it did not handle the API calls properly. This caused several problems in coding the transformation rules. For example, if these API methods were called within loops, the simulation program would crash. To solve this problem, a couple of modifications had to be done in the ZSI library, which should not have been done, since it strays from the scope of this project.

After dealing with all these problems the execution of transformation rules was implemented. Accompanying this report there is a flash file that shows a sample run on the test FSM model mentioned on the previous chapters. Also, below are pictures of a couple of steps in the simulation execution.

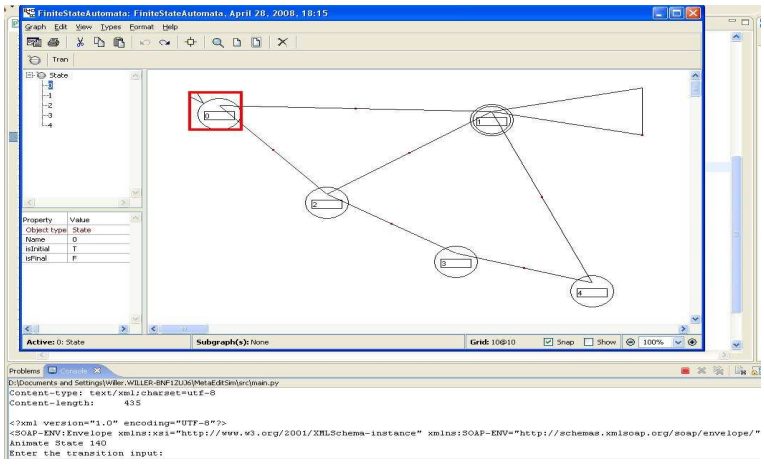

Fig. 6. The initial rule being executed.

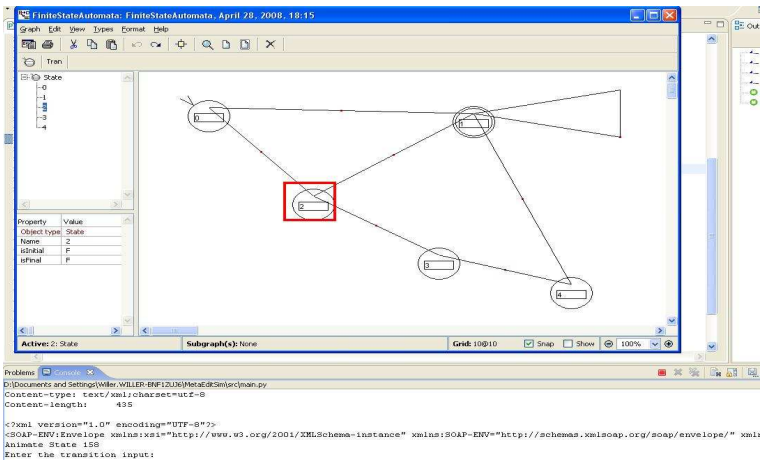

Fig. 7. The second rule being executed.

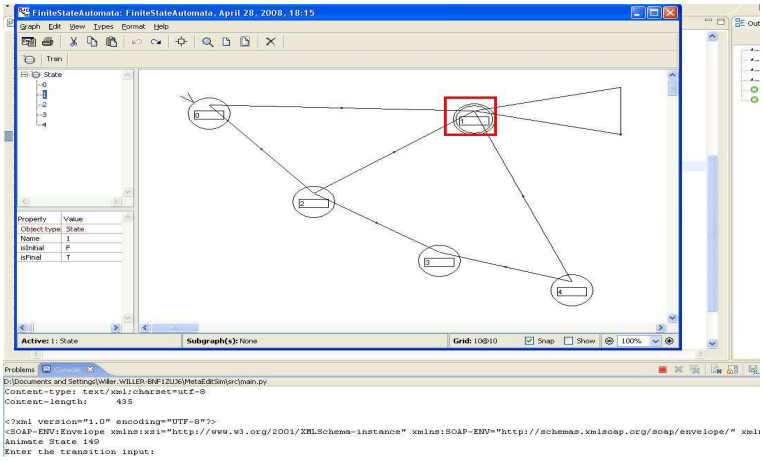

Fig. 8. StateTranstion being executed again.

6 Conclusions and Future Work

Even though MetaEdit+ does not have a way to execute transformation rules, its API does a great job of providing methods that allow a programmer to manipulate and extract model information. Also, MetaCASE's web service approach to its API provides enormous flexibility and accessibility to almost every programming language.

Although the web services approach provides access to any programming language, it also generates the problem of whether or not the SOAP libraries available for a language are capable of handling web services correctly.

As it was mentioned on the previous section, two libraries were used in this project. One wasn't able to parse WSDL files correctly, while the other could not execute the API's web services correctly, and had to be changed by the author. Therefore, the use of faulty libraries can render the MetaEdit+ API useless.

An approach to solve this was if MetaCASE could be provided such SOAP libraries, even though it is out of the work scope. It is important to note that these libraries did not present a good documentation on their respective websites. Also the error messages generated by the ZSI library were hard to comprehend, and did not help in the debugging process of the simulation.

Another problem with the Metaedit+ API is that its methods do not return direct access to objects and models. This causes the execution of programs using this API to slowdown considerably.

Finally, the documentation of the API is simplistic, and it only explains what methods do and nothing else. Also there is no support for error messages in their libraries which can cause problems for developers.

As for future work, it would be a great asset to have a generation code tool, like Eugene Syriani's MOTIF, to automatically generate transformation rules, created in AToM3, for MetaEdit+ containing the appropriate API calls.

References

1. Hans Vangheluwe, Juan de Lara: Meta-Models are Models too. Proceedings of the 2002 Winter Simulation Conference, pp. 1-9, 2002
2. Kelly, S., Lyytinen, K., and Rossi, M.: MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. Proceedings of CAiSE'96, 8th Intl. Conference on Advanced Information Systems Engineering, Lecture Notes in Computer Science 1080. Springer-Verlag, pp. 1-21, 1996
3. Juha-Pekka Tolvanen, Risto Pohjonen, Steven Kelly: SAdvanced Tooling for Domain-Specific Modeling: MetaEdit+. 7th OOPSLA workshop on Domain-Specific Modeling, pp. 1-8, 2007
4. Mark Pilgrim: Dive into Python. Chapter 12, pp 168-182, 2000, 2001, 2002, 2003, 2004. (http://www.diveintopython.org/toc/index.html)
5. Rich Salz, Christopher Blunck. ZSI: The Zolera SOAP Infrastructure. Chapters 2, 3, and 11, February 2005. (http://pywebsvcs.sourceforge.net/zsi.html)
6. MetaEdit+ 4.5 User's Manuals. MetaCase, March 2008. (http://www.metacase.com/support/45/manuals/index.html)
7. ATOM3 Documentation, McGill University. (http://atom3.cs.mcgill.ca/#doc)