AToMPM Solution for the IMDB Case Study

Hüseyin Ergin and Eugene Syriani

University of Alabama, Tuscaloosa AL, U.S.A.
`{hergin@crimson,esyriani@cs}.ua.edu`

Abstract. In this paper, we present an AToMPM solution for the IMDB case study of transformation tool contest 2014.

1 Introduction

AToMPM [1] allows one to model and execute model transformations. It provides a graphical user interface to define the metamodels of the input and output languages, define the transformation rules and their scheduling, and execute continuously or step-by-step transformations on given models.

The model transformation language of AToMPM is MoTif [2]. In MoTif , rules consist of a pre- and a post-condition. The pre-condition pattern determines the applicability of the rule and is usually defined with a left-hand side (LHS) and optional negative application conditions (NAC). The post-condition determines the result of the rule and is defined by a right-hand side (RHS) which must be satisfied after the rule is applied. Furthermore, any element in a rule in the LHS or RHS may be assigned to a pivot. It acts as a variable that can be referred to by other rules. To use a pivot, an element from the LHS or NAC can be bound to that pivot. The rule in Fig. 1 is a MoTif rule with a NAC, LHS, and RHS (from left to right). For the remaining of the paper, we have used a more concise notation to save space and annotate the rules as needed.

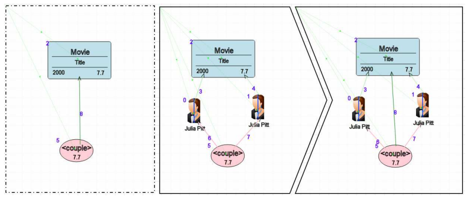

Fig. 1. MoTif rule as it appears in AToMPM .

The scheduling, or the control flow, describes the order in which the rules are executed. Each rule is represented by a rule block having three ports. Conceptually, a rule receives models via the input port at the top. If the rule is successfully applied, the resulting model is output from the success port at the bottom left. Otherwise, the model does not satisfy the pre-condition and the original model is output from the fail port at the bottom right. Fig. 2 depicts an example of control flow structure to schedule MoTif rules.

Some rule blocks are annotated in the scheduling, denoting a special behavior. The meaning of these rules are:

- *ARule*: is a regular "Atomic Rule" that is executed once on a single match. It has no annotation. (*e.g.*, *resetIterator* in Fig. 2)
- *FRule*: stands for "For all Rule". All matches are found first and then the rule is applied on all the matches. It is annotated with a letter 'F'. (*e.g.*, *computeAverage* in Fig. 4)
- *SRule*: stands for "Star Rule". It is a rule that is recursively applied on each match as long as matches are found. Therefore, the result of this rule is the accumulation of each application. It is annotated with an asterisk '*'. (*e.g.*, *createPositive* in Fig. 2)
- *QRule*: stands for "Query Rule". It is an *ARule* with no side effect since it does not have a RHS, but may still assign pivots. It is annotated with a question mark '?'. (*e.g.*, *findCouple* in Fig. 4)
- *CQRule*: stands for "Complex Query Rule". It is a nested *QRule* where a second query filters the result of the first one. It is annotated with two question marks '??'. (*e.g.*, *getOneCouple* and *notHighestRatingCouple* in Fig. 5)

This paper provides a solution to the IMDB model transformation case study, whose full description can be found at [3]. In Section 2, we provide the details about the solution. In Section 4, we summarize the results and conclude.

2 Solution

We have solved every task and extension of the case study in AToMPM . We used the same metamodel as given in the briefing document [3] with slight modifications. An integer *flag* variable is added to *Group* class to mark already processed groups while computing the top couples and top cliques. Also for the sake of simplicity, we have added a *movieNumber* attribute to *Group* class to hold the number of movies that group has. AToMPM does not have an iterator as a scheduling structure. For this reason, we have added an explicit *Iteration* class both to iterate on a rule and pass the value of the iterator to the rules to be used within. In the rules, *Iteration* class has a concrete syntax of a black rectangle and a text starting with "I" and having the current value and the limit of the iteration.

Each solution shows the rules on the left of the figure and the scheduling of these rules on the right.

2.1 Task 1: Generating Test Data

The first task is to generate the test data for the case. The rules and the scheduling of these rules are depicted in Fig. 2. The rules help to create a series of *Movie*s, *Actor*s and *Actress*es with the necessary relationships among each other. The rules mostly look like the original rules in the document, only with the addition of the *Iteration* class. The iterator makes the transformation run N times. This parameter can be set within the input model. We have an extra rule to reset the iterator before every use.

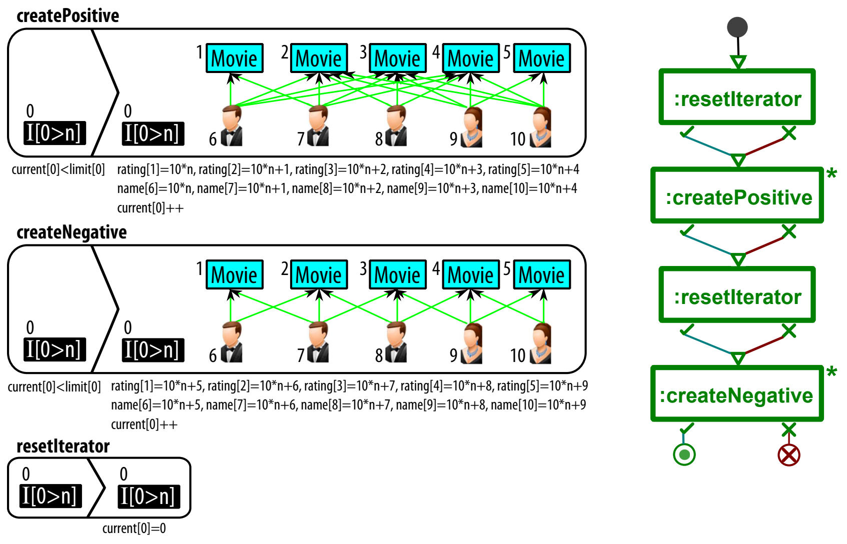
createPositive
Movie
I[0>n]
current[0]<limit[0]
rating[1]=10*n, rating[2]=10*n+1, rating[3]=10*n+2, rating[4]=10*n+3, rating[5]=10*n+4
name[6]=10*n, name[7]=10*n+1, name[8]=10*n+2, name[9]=10*n+3, name[10]=10*n+4
current[0]++
createNegative
current[0]<limit[0]
rating[1]=10*n+5, rating[2]=10*n+6, rating[3]=10*n+7, rating[4]=10*n+8, rating[5]=10*n+9
name[6]=10*n+5, name[7]=10*n+6, name[8]=10*n+7, name[9]=10*n+8, name[10]=10*n+9
current[0]++
resetIterator
current[0]=0
:resetIterator
:createPositive
:resetIterator
:createNegative

Fig. 2. Task 1 rules and scheduling.

2.2 Task 2: Finding Couples

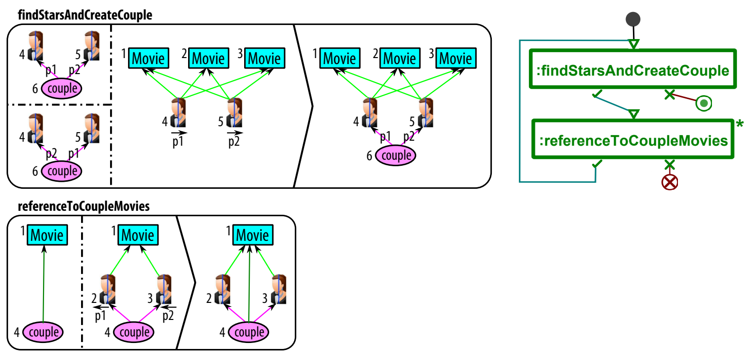

Fig. 3. Task 2 rules and scheduling.

This task aims to find two people who played in at least three movies together, create a *Couple* for them and reference to each movie they played in together. The rules and the scheduling of these rules are depicted in Fig. 3. The *findStarsAndCreateCouple* rule checks for two people that played in the same three movies. The rule will find the match if they have more than three movies too. Then a *Couple* class is created with a relation to each person. The NACs prevent to consider people already in couples. Since they can be either the *p1* or the *p2* of a couple, there are two NACs for each case. Pivots *p1* and *p2* are assigned to these people, so we can refer to these two persons in the following rule. The *referenceToCoupleMovies* rule creates a *commonMovies* relation from the newly created couple to each movie they played together, if not already referenced.

2.3 Task 3: Computing Average Rankings

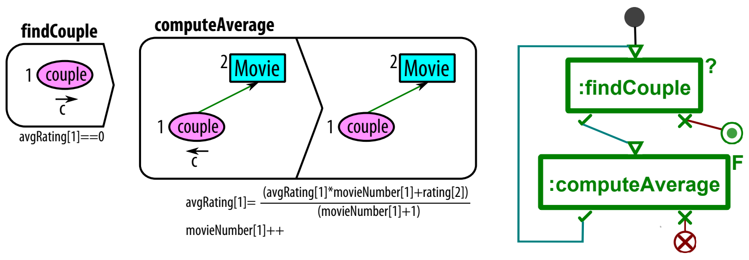

Fig. 4. Task 3 rules and scheduling.

This task is to compute the average rankings of each couple by using the *commonMovies* relation of the couples. The rules and the scheduling of these rules are depicted in Fig. 4. The *findCouple* rule finds a couple with *avgRating* zero, which means its average rating is not computed yet. It sets a pivot for this couple to be used in the next rule. Then, the *computeAverage* rule traverses all movies of this couple and computes moving average with increasing the movie number of the couple by one each time. The computation of the average is done in an intuitive way. First, the current average rating is multiplied by the current number of movies. Then, the rating of the current movie is added to this multiplication. Finally, the last number is divided by one more than the current movie number of the couple.

2.4 Extension Task 1: Compute Top-15 Couples

This task computes the top 15 couples and prints relevant information. The rules and the scheduling of these rules are depicted in Fig. 5. We use the iterator to compute the top N couples. This gives us the flexibility of setting the number of couples we want, directly within the model. The *resetIterator* rule resets the iterator before use. The *iterator* rule counts how many couples we want and it stops when we reach that number. Also we use the *current* attribute of this iterator to print the sequence number while printing information of a couple. In the scheduling, the *getOneCouple* and *notHighestRatingCouple* rules are put together inside a single CQRule (described in Section 1). This rule block finds a couple and eliminates it if it does not have the highest rating. It ends up with the highest rating couple at the end and sets a pivot to it. The *flag* attribute is used to mark the processed highest rating couple after printing the information. Then, the *printCoupleInformation* prints the necessary information to developer console, increases the current attribute of the iterator by one and sets the flag of the processed couple to 1.

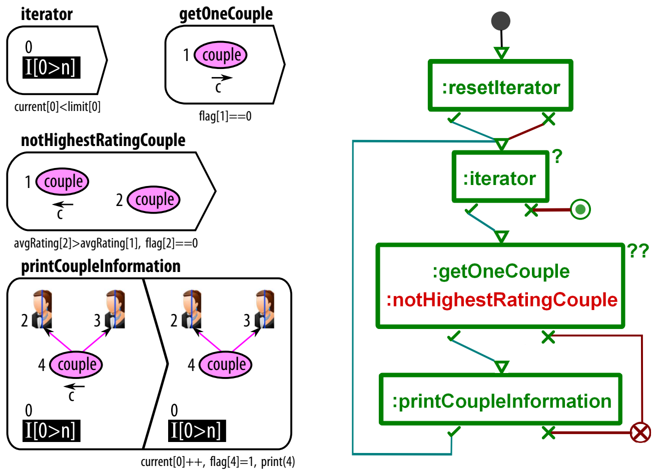

Fig. 5. Extension task 1 rules and scheduling.

The rules in the figure shows the solution for the top couples according to the average rating of their common movies. Solving the problem for the top couples according to the number of common movies is pretty easy. We add another rule, *notHighestMovieNumber*, which looks exactly like the *notHighestRatingCouple* rule, but it has a condition of $commonMovies[2] > commonMovies[1]$. The rest of the transformation is the same.

2.5 Extension Task 2: Finding Cliques

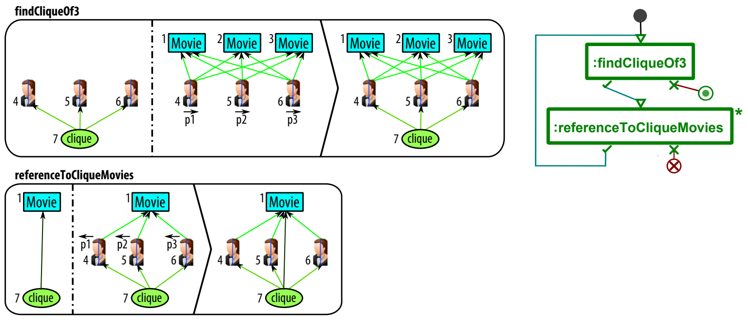

Fig. 6. Extension task 2 rules and scheduling.

This task aims at finding the cliques between people. A clique is a generalization of a couple with more than two people. The rules and the scheduling of these rules are depicted in Fig. 6. They are exactly the same as in task 2, but we changed the *Couple* to a *Clique* and added one more person.

The figure has the rules to find the cliques of three people. We did not show the rest of the rules for cliques of four and five, since they are exactly same copies with one and two more people added respectively.

2.6 Extension Task 3: Compute Average Rankings for Cliques

This task is to compute average ratings of each clique created in the previous extension task. The rules and the scheduling of these rules are depicted in Fig. 7. They are mostly the same as in task 3, which computes the average ratings for each couple. We just replaced the couple with a clique.

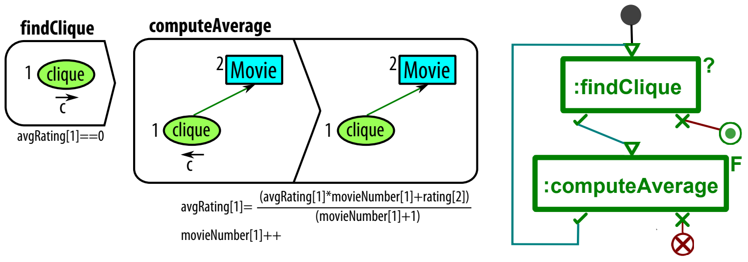

Fig. 7. Extension task 3 rules and scheduling.

2.7 Extension Task 4: Compute Top-15 Cliques

This tasks computes the top 15 cliques and prints information about them. The rules and the scheduling of these rules are depicted in Fig. 8. They are mostly like extension task 1, which computes the top couples. We have changed couple to clique to solve this task.

The rules solve this task by using the average rating of each clique. Adapting the problem to use the number of common movies is easy and just needs another rule as in extension task 1.

3 Performance

AToMPM depends on the network communication and this produces a bottleneck while computing the performance results. In this section, we have computed the performance results of the first task with the previous and native version of AToMPM , which is AToM^3 [4]. AToM^3 is implemented in python and executes faster for that reason. We give the performance results to be used as a reference. The results of *task 1: generating test data* with $N = 10$ for AToM^3 and AToMPM and for $N = 1000$ for AToM^3 can be seen below.

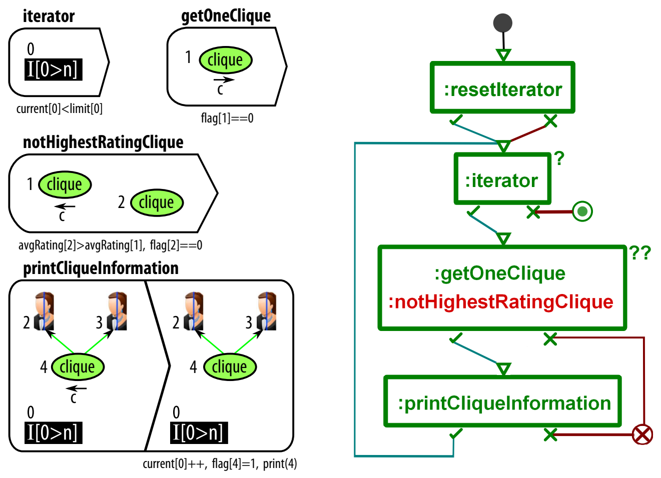

Fig. 8. Extension task 4 rules and scheduling.

- AToM^3 for $N = 10$ -> 0.25982144 seconds
- AToM^3 for $N = 1000$ -> 1171.11912448 seconds
- AToMPM for $N = 10$ -> 1.40664465 seconds

4 Conclusion

In this paper, we described the our solution of the IMDB case study using AToMPM . AToMPM heavily depends on graphical user interface and the handling of really large models is not possible in the current status. However, we are working on a headless environment and a new version of AToMPM to overcome these issues. Hence this solution focuses on the expressiveness and usability power of modeling and transforming in AToMPM , rather than its performance. In the SHARE machine, we put an appendix version of this paper to describe the steps to reproduce the test cases.

References

1. E. Syriani, H. Vangheluwe, R. Mannadiar, C. Hansen, S. Van Mierlo, and H. Ergin, “Atompm: A web-based modeling environment,” in *MODELS’13: Invited Talks, Demos, Posters, and ACM SRC. CEUR-WS.org*, 2013.
2. E. Syriani and H. Vangheluwe, “A Modular Timed Model Transformation Language,” *Journal on Software and Systems Modeling*, vol. 11, pp. 1–28, June 2011.
3. T. Horn, C. Krause, and M. Tichy, “The TTC 2014 Movie Database Case.” [Online]. Available: https://github.com/ckrause/ttc2014-imdb/raw/master/case_description.pdf
4. J. de Lara and H. Vangheluwe, “Atom^3: A tool for multi-formalism and meta-modelling,” in *FASE*, 2002, pp. 174–188.