The Movie Database Case: Solutions using Maude and the Maude-based e-Motions tool

Antonio Moreno-Delgado Francisco Durán
Dpto. Lenguajes y Ciencias de la Computación
University of Málaga, Spain
{amoreno,duran}@lcc.uma.es

The paper presents solutions for the TTC 2014 Movie Database Case, both in the e-Motions DSML and in the rewriting-logic formal language Maude. The DSMLs defined in e-Motions are automatically transformed into Maude specifications, which are then used for simulation and analysis purposes. e-Motions is a general purpose language, in which real-time languages may be modeled, with full support for OCL and other advanced features. The fact that the solutions given directly in Maude lack the overhead included by e-Motions to deal with all those extra features not needed in the current case study, makes these solutions much more efficient, and able to deal with bigger problems.

1 Introduction

Maude [1] is an executable formal specification language based on rewriting logic, which counts with a rich set of validation and verification tools, increasingly used as support to the development of UML, MDA, and OCL tools (see, e.g., [5]). Furthermore, Maude has demonstrated to be a good environment for rapid prototyping, and also for application development (see [1]).

Maude may be seen as a general framework where to develop model transformations. Durán, Vallecillo and others have used it to develop e-Motions [4], a tool that supports the definition and simulation of real-time Domain-Specific Modeling Languages (DSMLs). The e-Motions tool is a DSML and graphical framework developed for Eclipse that supports the specification, simulation, and formal analysis of real-time systems. It provides a way to graphically specify the dynamic behavior of DSMLs using their concrete syntax, making this task quite intuitive. Furthermore, e-Motions behavioral specifications are models too, so that they can be fully integrated in MDE processes.

In e-Motions, MOF metamodels are formalized in rewriting logic, providing a representation of the structural aspects of any modeling language with a MOF metamodel. Then, the behavior of such modeling language is specified as in-place transformation rules. Artifacts developed in e-Motions are automatically translated into Maude. e-Motions provides a very rich set of features, that enables the formal and precise definition of real-time DSMLs as models in a graphical and intuitive way. It makes use of an extension of in-place model transformation with a model of timed behavior and a mechanism to state action properties. The extension is defined in such a way that it avoids artificially modifying the DSML's metamodel to include time and action properties. Moreover, it supports attribute computations and ordered collections, which are specified by means of OCL expressions, thanks to mOdCL.[1] All these features make the language very expressive, but directly impacts its performance. To gain an idea of this impact, we provide below solutions to the proposed problems both in e-Motions and directly in Maude and compare them.

[1] mOdCL is available at `http://maude.lcc.uma.es/mOdCL`.

Submitted to:
Transformation Tool Contest 2014

The e-Motions system documentation and several examples are available at `http://atenea.lcc.uma.es/e-Motions`. The Maude web site is at `http://maude.cs.uiuc.edu`. The solution sources are at `http://github.com/antmordel/TTC14eMotions`.

e-Motions

The definition of a DSML typically comprises three tasks: (i) the definition of its abstract syntax, (ii) the definition of its concrete syntax and (iii) the specification of its behavior. In e-Motions the abstract syntax is defined by means of an Ecore metamodel, in which all the language concepts and the relations between them are specified. The concrete syntax is provided by defining the so-called Graphical Concrete Syntax (GCS). A GCS is a model (conforms the GCS metamodel) where an image is attached to each concept defined in the abstract syntax. Then, the behavior of a DSML is specified using visual graph-transformation rules. An e-Motions rule consists of a Left-Hand Side (LHS), a Right-Hand Side (RHS) and zero or more Negative Application Conditions (NACs). The LHS defines a (sub)-graph matching, optionally conditional. The RHS specifies a (sub)-graph replacement, which if the rule is applied, every object in the LHS that is not in the RHS is deleted, new objects in the RHS that are not in the LHS are created, and those objects whose attributes (or links) are changed are updated. NACs specify conditions or (sub)-graphs such that if there is a matching, the rule cannot be fired.

Rewriting Logic and Maude

Rewriting logic (RL) [3] is a logic of change that can naturally deal with state and with highly nondeterministic concurrent computations. In RL, the state space of a distributed system is specified as an algebraic data type in terms of an equational specification (Σ, E), where Σ is a signature of sorts (types) and operations, and E is a set of equational axioms. The dynamics of a system in RL is then specified by rewrite *rules* of the form $t \to t'$, where t and t' are Σ-terms. This rewriting happens modulo the equations E, describing in fact local transitions $[t]_E \to [t']_E$. These rules describe the local, concurrent transitions possible in the system, i.e. when a part of the system state fits the pattern t (modulo the equations E) then it can change to a new local state fitting pattern t'. Notice the potential of this type of rewriting, and the very high-level of abstraction at which systems may be specified, to perform, e.g., rewriting modulo associativity or associativity-commutativity.

Maude [1] is a wide spectrum programming language directly based on RL. Thus, Maude integrates an equational style of functional programming with RL computation. Maude also supports the modeling of object-based systems by providing sorts representing the essential concepts of object, message, and configuration. A configuration is a multiset of objects and messages (with the empty-syntax, associative-commutative, union operator `__`) that represents a possible system state.

Maude provides a whole formal environment where we can perform proofs of correctness of our solutions. Specifically, we have use the reachability analysis tool for performing checks on the correctness of our specification.

2 Solutions

We present two solutions for the different tasks, one graphical solution using e-Motions, and another one using directly Maude. Each task is solved by defining respective DSMLs, which share their abstract and concrete syntaxes. The abstract syntax used is the one provided in [2] — we will see below that some of the tasks have required extensions of this common syntax. The main differences between the DSMLs

defined for the different tasks is in their concrete behaviors describing what need to be done in each case, that is, the rewrite rules defining the behavior depends on the concrete task and its solution.

Although the expressiveness of e-Motions is very welcome in complex problems, thanks to its capabilities to express problems visually, very intuitively, and in a language very close to the problem domain, the overhead to be paid in cases like the ones at hand is too high. Specifically, the generality provided by its support for OCL expressions and time requirements, makes that the Maude code generated by the e-Motions tool is not as time performant as we would like. However, the general purpose rewrite-modulo engine at the core of Maude may also be used as a transformation language. Thus, together with the e-Motions solution we present an optimized Maude solution for each task.

As we will see below, the Maude version of the transformation closely follows the transformations provided in e-Motions, were all rewrite rules are *instantaneous* and expressions are solved directly by Maude built-in types instead of by the OCL interpreter. Indeed, for problems as simple as the ones at hand, we will see that the representation distance between Maude and e-Motions to the problem domain would be very small, making both solutions very appropriate. Although a more in depth analysis of the problem at hand would most probably have allowed us to even improve the numbers obtained, we have preferred to keep the specification clear and intuitive.

Task 1

Task 1 comprises the generation of synthetic models (conforming the movie database metamodel [2]) from an input parameter $N \geq 0$. We first present an e-Motions solution and then a Maude solution.

Firstly, following an e-Motions based approach, we define the abstract and concrete syntax and the behavior of our so-called *Task 1 DSML*. Taking a parameter *N* as input model, *Task 1 DSML* generates a model containing synthetic data. As it has been introduced in Sect. 1, the abstract syntax of a DSML is given in e-Motions by means of an Ecore metamodel. Since we model the solution of the task as a model that evolves until reaching its final solution, we take as metamodel the one provided in [2], which we call *Movies MM*, extended with a `Parameter` concept. The class `Parameter` has two integer attributes, which represent positive graphs and negative graphs, respectively, for the generation following Henshin graphs [2]. For the concrete syntax, Fig. 4 in Appendix A shows how an image has been attached to each concept modeled in the Movies MM.

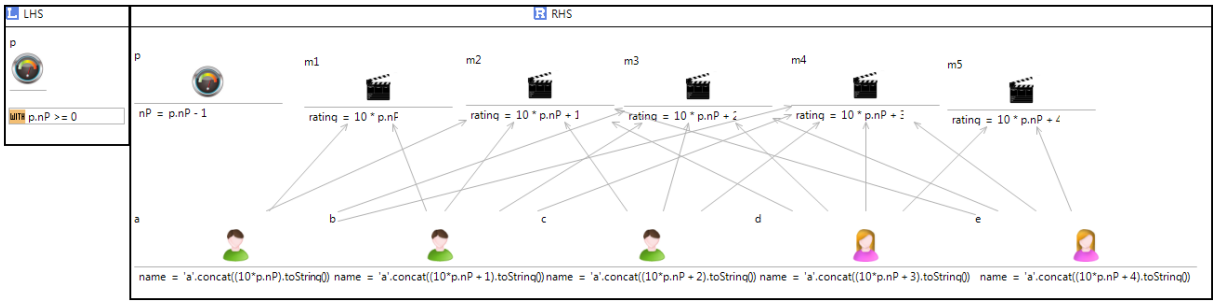
The behavior of this *Task 1 DSML* is then given by means of two in-place transformation rules: `createPositive` and `createNegative`. Fig. 1(a) shows the `createPositive` rule, which takes an object `p` of type *Parameter*, with *nP* attribute greater or equal than 0, and produces synthetic data conforming to the Henshin rules. Fig. 1(b) shows the `createNegative` rule, which is analogously defined.

Note that this solution is really close to the problem specification in [2]. Fig. 1, and Fig. 2 in the case description [2], specifying the data generation, are almost the same. This demonstrates how close the solution by e-Motions is to the problem domain, and how convenient its graphical facilities are.

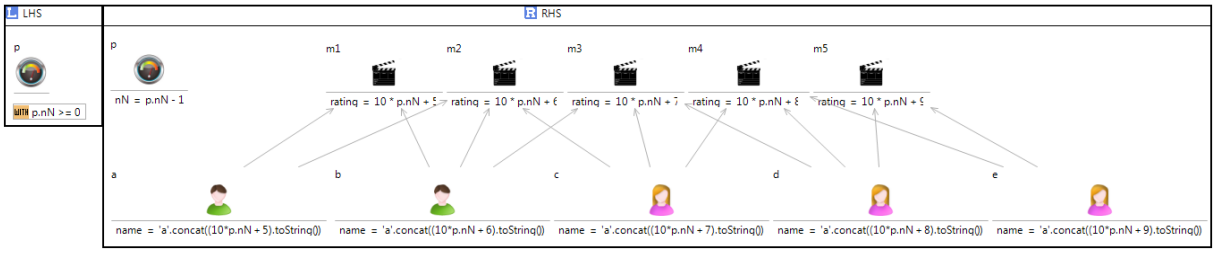
Our Maude-based solution for Task 1 consists of an object-based Maude specification, which matches very closely the e-Motions solution. See Appendix B) for the Maude specification of rule `createPositive` and for a comparison of the number of rewrites and execution times for both solutions.

Task 2

Task 2 consists in finding all ‘couples’ from a given model, given that two persons are a ‘couple’ if they played together in at least three movies [2]. Couples are to be obtained from the model obtained in Task 1.



(a) The `createPositive` rule.



(b) The `createNegative` rule.

Figure 1: Task 1 rules.

The e-Motions-based solution for this task is implemented with one single rule, `createCouple`, shown in Fig. 2. `Person` objects are shown using square shapes because `Person` is an abstract class and it does not have attached image. The `createCouple` rule models the creation of a couple by taking two persons and generating a couple with them. The rule has two conditions: a positive condition stating that *"the number of movies in the intersection between the movies of `per1` and `per2` is greater or equal than 3"*; and a negative condition, `coupleHasNotBeenCreated`, requiring that the couple does not exists yet.

See Appendix C for an alternative specification of the e-Motions solution, in which we reduce the number of candidate matchings, Maude specifications of the solution, and a comparison of the results.

Task 3

Given a model with couples already created, Task 3 consists in calculating the average rating of

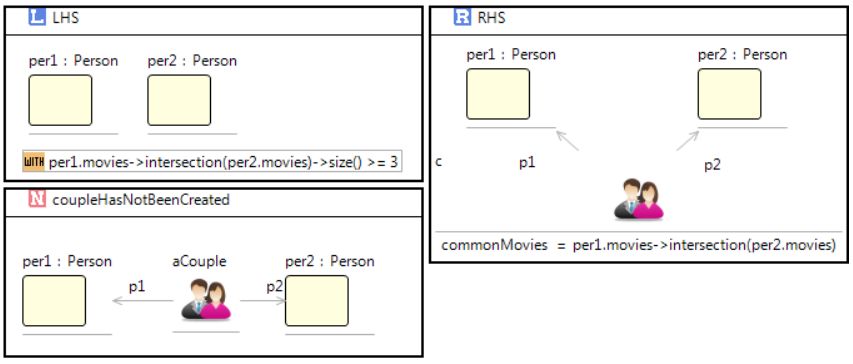


Figure 2: e-Motions rule `createCouple`.

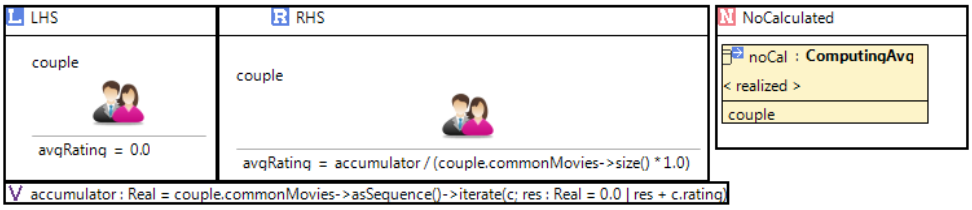


Figure 3: `computingAvgRating` rule.

shared movies for each of these couples.

The e-Motions-based solution consists in one single rule, shown in Fig. 3, in which the average is calculated only once for each couple. Notice the use of an action in the NAC of the rule to state that the value has not already been calculated.

See Appendix D for the Maude counterpart, and a comparison of the number of rewrites and execution times for the solutions.

3 Conclusions

We have presented solutions for the TTC 2014 Movie Database Case both in the e-Motions DSML and in the rewriting-logic formal language Maude.

e-Motions provides a very rich set of features, that enables the formal and precise definition of real-time DSMLs as models in a graphical and intuitive way. It makes use of an extension of in-place model transformation with a model of timed behavior and a mechanism to state action properties. The extension is defined in such a way that it avoids artificially modifying the DSML's metamodel to include time and action properties. Moreover, it supports attribute computations and ordered collections, which are specified by means of OCL expressions. All these features makes the language very expressive, but directly impact on performance.

The Maude solutions presented are also very intuitive and simple. The fact that the solutions given directly in Maude lack the overhead included by e-Motions to deal with all those features it provides that are not needed in the current case study, makes the solutions given much more efficient, and able to deal with bigger problems.

Acknowledgments. This work is partially funded by Projects TIN2012-35669 and TIN2011-23795.

References

[1] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer & C. Talcott (2007): *All About Maude - A High-Performance Logical Framework* *LNCS* 4350, Springer.

[2] T. Horn, C. Krause & M. Ticky: *The TTC 2014 Movie Database Case*. Available at TTC14 web site.

[3] J. Meseguer (1992): *Conditioned Rewriting Logic as a Unifed Model of Concurrency*. *TCS* 96(1): 73–155.

[4] J. E. Rivera, F. Durán & A. Vallecillo (2010): *On the Behavioral Semantics of Real-Time Domain Specific Visual Languages*. In: *WRLA*: 174–190.

[5] J. R. Romero, J. E. Rivera, F. Durán & A. Vallecillo (2007): *Formal and Tool Support for Model Driven Engineering with Maude.* *Journal of Object Technology* 6(9): 187–207.

A Figures


(a) Actor.


(b) Actress.


(c) Movie.


(d) Couple.


(e) Parameter.

Figure 4: Concrete syntax for *Movies* MM*.

B Maude listings and results for Task 1

As in the e-Motions solution for Task 1, the Maude solution has two rewrite rules: `createPositive` and `createNegative`. Listing 1 shows the `createPositive` Maude rule, which takes the message `createPositive(s(N:Nat))` and returns a configuration conforming the Henshin specification [2]. A similar rule generates the negative cases. Notice that the Maude solution is very much like the e-Motions solution. In fact, the former could be seen as the textual version of the latter.

The execution performance for both solutions is shown in Table 1, which shows the number of rewrites and execution times for both solutions. As explained above, the execution times for the Maude specification obtained from the e-Motions definition grows very quickly. Notice that, although the number of rewrites grows linearly with respect to N, the time is exponential due to the infrastructure to deal with all the extra features in e-Motions. However, notice how the number of rewrites for the Maude solution grows linearly as well, but in this case the execution times grow more slowly, being able to handle problems of much bigger sizes.

Listing 1: `createPositive` Maude rule.

```
rl [createPositive] :
  createPositive(s(N))
  freshOid(N')
=>
  createPositive(N)
  < N'     : Movie | rating: (10.0 * float(N)) >
  < N' + 1 : Movie | rating: (10.0 * float(N) + 1.0) >
  < N' + 2 : Movie | rating: (10.0 * float(N) + 2.0) >
  < N' + 3 : Movie | rating: (10.0 * float(N) + 3.0) >
  < N' + 4 : Movie | rating: (10.0 * float(N) + 4.0) >

  < N' + 5 : Actor | name: ("a" + string(10 * N, 10)),
                     movies: (N', N' + 1, N' + 2, N' + 3) >
  < N' + 6 : Actor | name: ("a" + string(10 * N + 1, 10)),
                     movies: (N', N' + 1, N' + 2) >
  < N' + 7 : Actor | name: ("a" + string(10 * N + 2, 10)),
                     movies: (N' + 1, N' + 2, N' + 3) >
  < N' + 8 : Actress | name: ("a" + string(10 * N + 3, 10)),
                     movies: (N' + 1, N' + 2, N' + 3, N' + 4) >
  < N' + 9 : Actress | name: ("a" + string(10 * N + 4, 10)),
                     movies: (N' + 1, N' + 2, N' + 3, N' + 4) >
  freshOid(N' + 10) .
```

	e-Motions		Maude	
N	Time (s)	# Rewrites	Time (s)	# Rewrites
1			0.0	67
2	0.0	4,910	0.0	133
10	0.0	24,334	0.0	661
20	0.0	48,614	0.0	1321
100	0.6	242,854	0.0	6601
1000	55.7	2,428,054	1.7	66,001
2000	395.0	4,856,054	11.8	132,001
3000			31.5	198,001
4000			40.8	264,001
5000			65.8	330,001
6000			96.8	396,001
7000			133.4	462,001
8000			175.8	528,001
9000			224.5	594,001
10000			227.9	660,001
11000			337.4	726,001

Table 1: Times for the e-Motions and Maude solutions to Task 1.

C Maude listings and results for Task 2

Although very intuitive and simple, the e-Motions solution presented in Section 2 for Task 2, is computationally very expensive. Notice that the number of matchings in the LHS of the rule is quadratic on the input size, leaving all the task to the evaluation of the conditions to accept or discard the couples. We have implemented another solution in which we limit (although we do not reduce the problem complexity) the number of matchings using a very simple algorithm: For each person, we iterate on the rest of persons looking for couples. With this algorithm, the number of persons to match as candidate couples decreases significantly.

As for the Maude-based solution, we have specified both solutions. Both solutions match very closely their e-Motions counterparts. The Maude specification of the first alternative solution to Task 1 is shown in Listing 2. The rules takes two persons and creates a new couple if they share three movies and such couple has not been previously created. Some numbers for its execution are shown in Table 2.

However, while for the Maude-based solution we get better results with the enhanced solution, for the e-Motions one we get even worse time executions. This is due to the high overhead included in e-Motions by each additional rule, since the enhanced solution has more rules that the naive one.

Listing 2: `createCouples` Maude rule.

```
crl [findCouples] :
 { freshOid(N) findCouples
   < O1 : V1:Person | movies : MS1, Atts1 >
   < O2 : V2:Person | movies : MS2, Atts2 >
   Conf }
=>
 { freshOid(s(N)) findCouples
   < O1 : V1:Person | movies : MS1, Atts1 >
   < O2 : V2:Person | movies : MS2, Atts2 >
   < N : Couple |
           commonMovies : (intersection((MS1), (MS2))),
           p1 : O1, p2 : O2 >
   Conf }
if | intersection((MS1), (MS2)) | >= 3
/\ not coupleInConf(C, Conf) .
```

N	Time (s)	# Rewrites
1	0.0	8,680
5	0.5	1,343,000
10	5.0	11,020,000
20	66.3	89,276,000
30	314.0	302,568,000

Table 2: Maude times for Task 2 First Version.

N	Time (s)	# Rewrites
1	0.0	831
10	0.1	82,983
100	8.9	8,299,803
200	68.3	33,199,603
300	242.9	74,699,403
400	640.1	132,799,203

Table 3: Maude times for Task 2 Second Version.

D Maude listings and results for Task 3

The Maude rule specifying the solution of this task is shown in Listing 3. The number of rewrites and execution times of the e-Motions solution for Task 3 in Section 2 for $N = 2, 10$ are shown in Table 4.

Listing 3: Maude rule for Task 3 solution.

```
crl [avgRating] :
  { < M : Couple | commonMovies : MovieSet,
       avgRating : 0.0,
       Atts1 >
    couplesCalculated(Couples)
    C
  }
=>
  { < M : Couple | commonMovies : MovieSet,
       avgRating : sumAllRatings(MovieSet, C)
                     / float(| MovieSet |),
       Atts1 >
    couplesCalculated((M, Couples))
    C
  }
if not(M in Couples) .
```

Table 5 shows the number of rewrites and execution times for the Maude solution for problems of sizes 100, 200, 300, and 400.

N	Time (s)	# Rewrites
2	0.0	4,527
10	2.1	891,432

Table 4: e-Motions times for Task 3.

N	Time (s)	# Rewrites
100	1.5	21,800
200	6.3	43,600
300	14.9	65,400
400	29.7	87,200

Table 5: Maude times for Task 3.