Using MoTif for the AntWorld Simulation Tool Contest

Eugene Syriani and Hans Vangheluwe

McGill University, School of Computer Science, Montréal, Canada,
{esyria,hv}@cs.mcgill.ca

Abstract. The AntWorld Simulation case study proposed at the GraBaTs 2008 workshop tool contest reflects the ability of graph transformation tools to employ local search. This paper provides a solution using the Modular Timed graph transformation language MoTif.

1 Introduction

The Modular Timed Graph Transformation language (MoTif) allows to model and execute model transformations. On the one hand, it provides a graphical user interface for the description of the graph transformation rules in a declarative way and on the other hand, a modelling environment to define the control structure of the transformation. For more information on MoTif, the reader is suggested to read [1] where the essence of the language is overviewed and to [2] where it was used for simulation-based design.

This paper is a solution to case no. 2 (AntWorld Simulation case study) of the GraBaTs 2008 tool contest using MoTif as the graph transformation language. The full description of the case study can be found at [3]. This case study stresses local rule application and we show how this can be done with MoTif.

In Section 2, the essential parts of the solution as well as performance results are presented. In Section 3, we summarize this work and outline some advantages of this approach.

2 Solution

In our approach we define a domain-specific modelling environment for the AntWorld formalism in the modelling tool AToM^3 [4]. Hence a meta-model needs to be defined.

2.1 The Meta-Model

As shown in Figure 1, the formalism consists of Ants and GridNodes. An Ant element can go on a GridNode which can also be a Hill. GridNodes can hold pheromones and food parts. Ants can be in "carry mode". The grid nodes are connected in circles centred at the hill in a very specific way. Our solution realizes that by differentiating between connections in the same circle and to the next circle for neighbouring grid nodes. Furthermore, different strategies for the generation of the grid are used depending if it is a node along the two main axis. This is why we distinguish between main axis nodes and the other grid nodes. A node counter is also needed to decide which generated node food will hold food parts. Using AToM^3 as a modelling environment provides concrete syntax for each meta-model element as illustrated in the rules specification.

2.2 The Transformation Rules

The graph transformation rules in MoTif consists of three parts: the left-hand side (LHS), the right-hand side (RHS) and the negative application condition (NAC). For example, the *ConnectNodesInSameCircle*

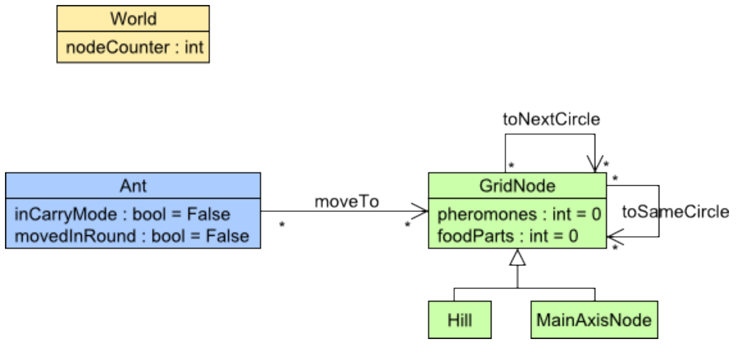


Fig. 1. AntWorld Meta-Model

rule (figure 2) states that whenever two neighbouring nodes[1] (N and M) are on the same circle and are each linked to a GridNode (labelled respectively 1 and 2) on the next circle, a connection toSameCircle must be drawn between these latter nodes in the same direction as the N and M are connected. However GridNode 1 may not be the source of a link to a node on the same circle nor GridNode be the target of a similar link. Also, node N must be bound by the pivot that this rule receives and M will be the pivot of the next rule. This internal dependencies between rules is the essence of how local rule application is achieved in MoTif. For this particular case, the next rule that will be executed is again *ConnectNodesInSameCircle* (see the control structure in the next sub-section). This makes the connecting step between generated nodes ordered in a clockwise direction. If the pivot nodes were not provided, every legal pair of nodes will be connected randomly. Therefore we not only apply rules locally for ants but also at the level of the grid generation, as we will see in the next sub-section.

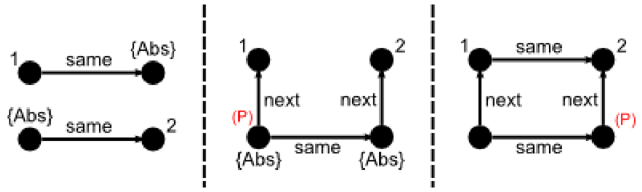


Fig. 2. The *ConnectNodesInSameCircle* rule

The complete list of the rules used for the simulation is provided in Appendix A.
Once all the rules are defined in AToM^3, we compile them to be callable by the MoTif simulator.

2.3 The Control Structure

We have seen how the rules can be sequenced implicitly by pivot passing. At the control structure level, we express the order of execution of the rules by means of sequencing, branching, looping or in parallel. MoTif provides some primitive blocks to achieve that. A *CRule* block encapsulates sub-rule blocks in a

[1] We use the term nodes in the general sense (i.e.: including GridNode, MainAxisNode and Hill) because subtype matching is toggled when an element is flagged with the {Abs} label, short for abstract.

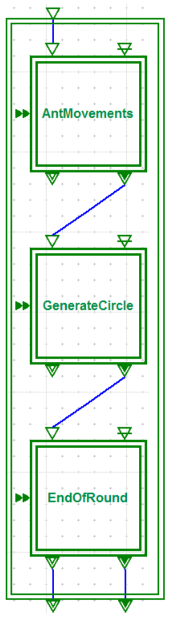
highly modular way interfacing via some inports (*GraphIn* at the top left, *AbortIn* at the top right and *Next* on the side) and outports (*SuccessOut* on the bottom left and *FailOut* on the bottom right). An *ARule* refers to a compiled rule and applies the rule atomically. A rule application is done in two phases: first the matching (where all the possible matching are found) and then the transformation on one or more matches. When an outport has a small round on the summit of its triangular shape, it means that along with the transformed graph, in the case of success, the pivot information is sent. In the case of of failure the received pivot is passed to the next rule. The other block used in this transformation system is the Synchronizer which allows the synchronization of matches found in parallel on different rules.
At each round, the graph (or input model) is received by the *GGRule CRule* (Figure 3(a)). First in a priorities fashion, first the *AntMovements* rules are applied, then the *GenerateCycle* rules and finally the *EndOfRound* rules.
We will give a high-level description of the transformation without going in the details since figures 3(b), 4(a) and 4(b) are straight forward. For the complete specification, the full listing is provided in Appendix B.
First we look for an ant to grab a food part, if it succeeds this ant moves one step towards the hill. Then the graph is sent back to the *GrabFood ARule* but via the Next port to only choose the next matching and apply the transformation.
When no more ant can move towards the hill an ant in carry mode found on the hill drops its food part and goes in search mode. When in search mode, the ant first tries to follow a pheromone trail or else moves randomly in any possible direction. That is achieved by the four *ARules* connected to the Synchronizer which find their corresponding matches in parallel and only one is chosen randomly to apply its transformation. When no ant was provided by *DropFood*, all ants (iteratively randomly chosen) in search mode which have not moved yet make a move randomly.
When there is no more ant left to be moved, we check if an ant has reached the outermost circle and start the generation of the nodes of the next circle from that node in clockwise order. In the meantime, at every tenth node, some food is created on it.
At the end of each round, for each food part on the hill one ant element is created. Note that *AntBirth* is an *FRule* where all matches are transformed at the same time. This is safe since no two matchings can be critical pairs. Similarly, all pheromones are evaporated and finally the move flag on each ant is reset.

2.4 Simulation

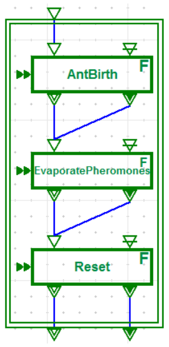
MoTif is compiled into a discrete event simulation engine. It can be ran until a specific condition is reached or for some number of steps (one step is consumed when the higher *CRule* in the hierarchy level sends out a graph). In our case we ran the simulation with no limit and some performance measurements have been collected at the end of each round. Table 1 shows some results per round, the number of circles present on the grid, the total number of nodes, the number of food parts present on the grid the total number of ants alive. And to each round we show how long the transformation step took in seconds. So the 165^{th} round, took 7 minutes and 18 seconds while the generation of the 13^{th} circle (100 nodes) took about 48 minutes. The total execution time was $23,890$ seconds. Also, on average over the first 165 rounds the transformation time is about 144 seconds and if we do not consider the time consumed by the *GenerateCycle* block, the average is 104 seconds.
These measurements were taken on a Windows Vista machine with a Intel Core 2 Duo CPU with 1.5 GHz of RAM.

3 Conclusion

The controlled graph rewriting language, MoTif, was able to fullfill the requirements of the published case study. Given an input model (in our case compiled from AToM^3), the transformation is successfully

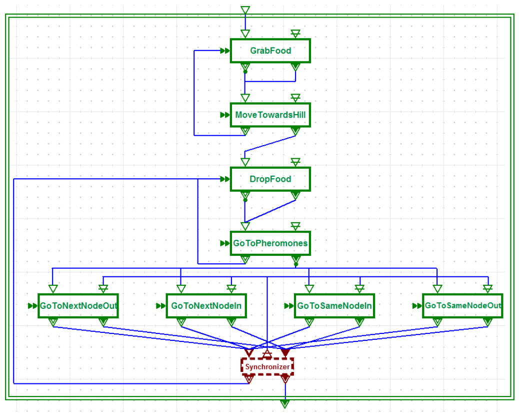


(a) The *GGRule CRule*

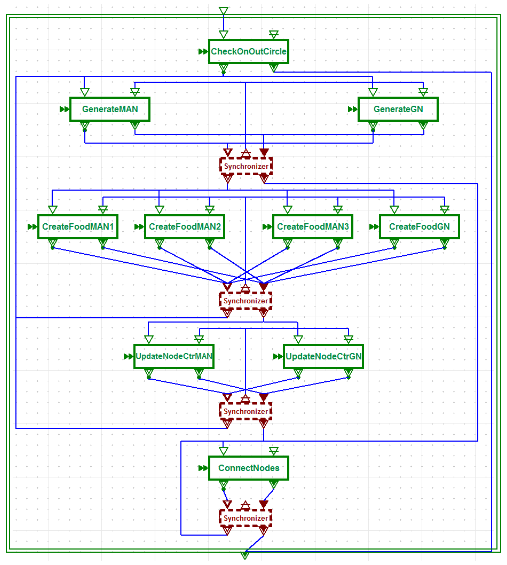


(b) The *EndOfRound CRule*

Fig. 3. Some graph transformation control structure



(a) The *AntMovements CRule*



(b) The *GenerateCycle CRule*

Fig. 4. Some graph transformation control structure

Round	Circles	Nodes	Food	Ants	Time (sec)	Round	Circles	Nodes	Food	Ants	Time (sec)
1	2	16	0	8	0	43	7	196	1724	34	17
8	4	64	500	8	0	44	7	196	1713	35	7
9	5	100	900	8	15	55	7	196	1651	46	10
10	5	100	900	8	1	56	8	256	2237	47	207
11	5	100	900	8	1	65	9	324	2891	56	48
12	5	100	900	8	1	73	10	400	3640	64	698
13	5	100	899	8	0	103	10	400	3423	94	74
14	6	144	1297	8	39	104	11	484	4214	95	1218
15	6	144	1297	8	0	105	12	576	5107	96	1924
32	6	144	1267	23	1	106	12	576	5099	97	117
33	6	144	1264	24	2	107	13	676	6093	98	2881
38	6	144	1243	29	3	165	13	676	5523	156	438

Table 1. Performance Measurements

applied using local application of the rules. This was done by binding graph element to given pivot information as well as pivot passing at the structure level. A visual modelling environment in concrete syntax is provided for the specification of the rules and the description of the flow of transformation.
Some remarks can be made regarding the design of the solution. Although the graphs used have directed edges, the transformation process gives the semantics of undirected edges, which is needed for the movement of the ants. However having directed edges simplified some of the rules for the node generation.
All random choices in MoTif are repeatable for simulation purposes. That is every time a new simulation is run, the same random choices will be performed each time. This is important for repeatable experiments.
This case study does not require any use of time except for performance measurements. MoTif is a timed language allowing the modeller to specify how long a rule will spend in the matching phase. In our design, we have set all the rules to spend 1 time unit. A real-time execution could have also been used for simulating the AntWorld problem.
Soon a pure parallel implementation of MoTif will be available which will increase the performance significantly. Also MoTif will be extended to link back to AToM^3 in order to provide visual simulation in a graphical user interface.

References

1. E. Syriani and H. Vangheluwe, "Programmed graph rewriting with DEVS," in *3rd International Symposium Applications of Graph Transformations with Industrial Relevance (AGTIVE 2007)*, ser. Lecture Notes In Computer Science, M. Nagl and A. Schür, Eds. Kassel: Springer-Verlag, October 2007.
2. ——, "Programmed graph rewriting with time for simulation-based design," in *International Conference on Model Transformation (ICMT 2008)*, ser. Lecture Notes in Computer Science, A. Pierantonio, A. Vallecillo, J. Bézivin, and J. Gray, Eds., vol. 5063. Zürich: Springer-Verlag, July 2008, pp. 91–106.
3. [Online]. Available: http://www.fots.ua.ac.be/events/grabats2008/
4. J. de Lara and H. Vangheluwe, "AToM^3: A tool for multi-formalism and meta-modelling," in *5th International Conference on Fundamental Approaches to Software Engineering*, ser. Lecture Notes in Computer Science, R.-D. Kutsche and H. Weber, Eds., vol. 2306. Grenoble: Springer, April 2002, pp. 174–188.