AntWorld Simulation Case Study Modeled by Tiger

Claudia Ermel, Enrico Biermann
Email: tigerprj@cs.tu-berlin.de
URL: http://tfs.cs.tu-berlin.de/tigerprj

Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany

Abstract. We model the AntWorld simulation case study using the Tiger Tool which supports rule-based specification of visual languages and generation of visual editors based on the Eclipse Graphical Editor Framework.

1 Introduction

Domain specific modeling languages are of growing importance for software and system development. Meta tools are needed to support the rapid development of domain-specific visual editors. A visual language (VL) definition based on a meta model in combination with a rule-based specification of editor commands is used in TIGER *(Transformation-based Generation of Environments)* to generate a corresponding visual editor.

TIGER combines the advantages of precise VL specification techniques using graph transformation concepts with sophisticated graphical editor development features offered by the Eclipse Graphical Editing Framework (GEF) [1]. Using graph transformation at the abstract syntax level, an editor command is modeled in a rule-based way. The application of such syntax rules to the underlying syntax graph of a diagram is performed by the graph transformation engine AGG [2]. TIGER extends AGG by means for concrete syntax definition. From the VL definition, Java source code is generated, implementing an ECLIPSE visual editor plug-in based on GEF. Thus, the generated editors appear in a timely fashion, conforming to the ECLIPSE standard for graphical tool environments. For the AntWorld case study, graph rules have been used not only for editing the visual AntWorld, but also for modeling the behavior of the ants, and the expansion of the grid and the ant population. Since Tiger is an editor generator, where graph rules are translated to palette or context menu entries, no control structures are supported. Nevertheless, in this paper we suggest a control structure for the rules which is essential to model the case study, but this control structure has to be implemented by extending the code of the generated editor.

2 Antworld Type Graph

Fig. 1 shows the abstract syntax type graph for our *AntWorld* visual language. We model *Ants*, *Fields*, *Food* and *Pheromones* as symbol types. *Fields* are further divided into *Hill*, *NormalFields* and *MainAxisFields*. The *Hill* field is the center of the *AntWorld* grid, *MainAxisFields* form the main four axis in the grid starting from the hill. All other fields are *NormalFields*. The FoodFieldCounter symbol type is a global counter to enable a regular distribution of food on the grid.

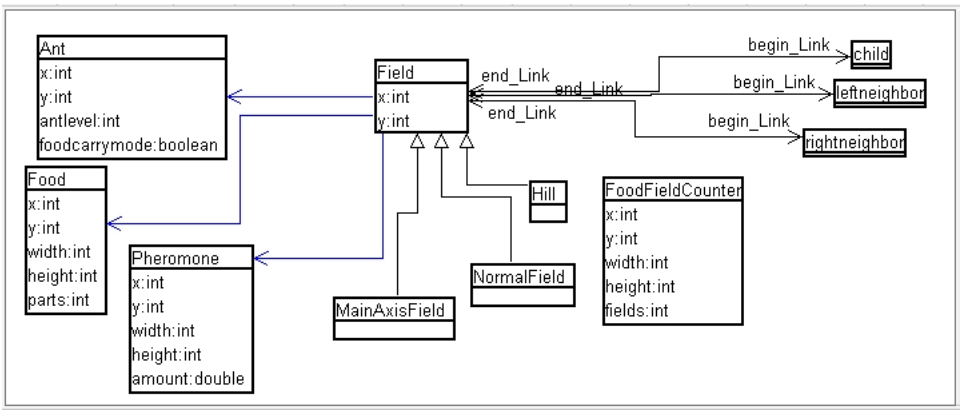

Fig. 1. AntWorld Type Graph

3 Visual Appearance

Tiger allows to define figures composed of rectangles, ellipses and polygons to be assigned to symbol types. Recently, Tiger was extended by the possibility to define figures contained in other figures. This can be seen in the type graph in Fig. 1 by the containment arcs (the blue arcs) from the container type *Field* to the contents types *Ant*, *Food* and *Pheromone*. Contents symbols are depicted in their respective container symbols and are moved together with their containers. Fig. 2 shows the start graph of the AntWorld simulation, where two circles around the hill exist, and eight ants are on the hill. Main axis fields are depicted as blue circles, normal fields are white.

4 Rules

4.1 Moving Ants

We have six rules for ant movement. In search mode, an ant can move up or down or along a circle as long as there are no pheromones on any child field

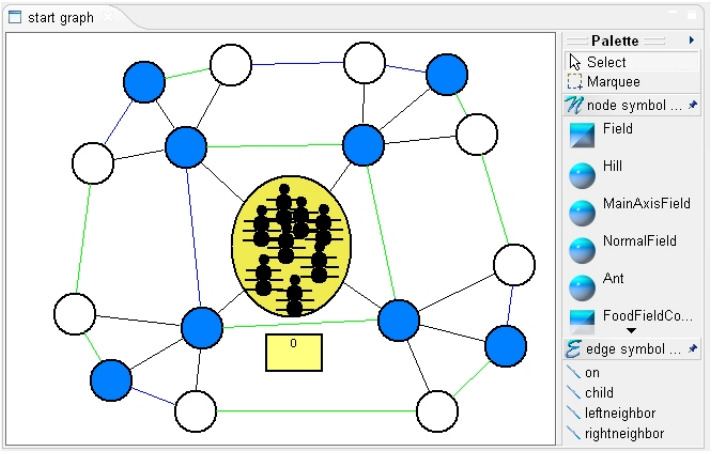

Fig. 2. AntWorld Start Graph

from the ant's current position (see e.g. rule MoveDown in Fig. 3. If there are pheromones on any child field, a rule FollowPheroTrail is used instead of the other movement rules to guide the ant to a food place.
If the ant already picked up some food, it will use one of the two CarryFood rules: CarryFoodRefreshPheromone refreshes an already existing pheromone trail while the ant moves toward the hill; rule CarryFoodDropNewPheromone creates a new pheromone path.

4.2 AntWorld Management

AntWorld management consists of expanding the world if an ant moves towards the outer edge, creation of new food supplies on the outer ring, and decay or deletion of old pheromone trails. The world is expanded by three rules: ExtendMainAxis (see Fig.4), ExtendNormal and ConnectNeighbors to connect newly generated outer nodes.
While expanding the world, the number of newly created fields is stored in the symbol FoodFieldCounter, and afterwards new food supplies (100 parts) are created on the outer ring (see rule PutFirstFoodOnCircle and AddFoodToNewCircle in Fig. 4).
As the last step, every pheromone occurence is reduced by a factor of 0.95 (rule DecayPheromone). If any pheromone is removed below 9, it is removed completely.

4.3 Food Transportation

If an ant comes to a field containing food, it will pick up one part of the food and start moving towards the hill using movement rules. Once on top of the hill it will drop the food (rule AntDropsFoodOnHill), changes its foodcarrymode attribute from true to false and creates a new ant.

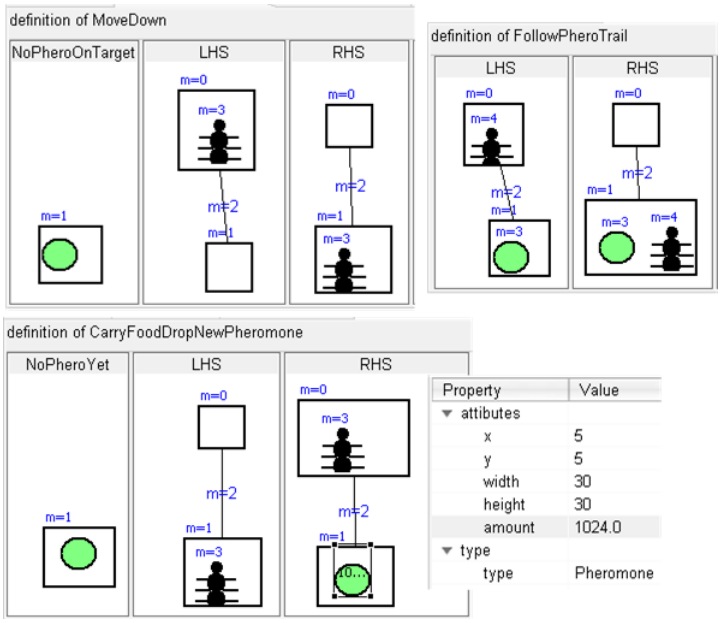

Fig. 3. Some Rules for Moving Ants

5 Control structure

For the rule control flow we extend the generated Java code of the GEF based visual editor to define application units for different task groups as defined in the previous section.
In our case, a round consists of

- Ant movement,
- AntWorld management
- food transportation and ant creation

In the unit for ant movement, we allow each ant exactly one rule application. If the ant can carry food or follow a trail, it will do it. If not, it is in search mode and will choose one of the possible matches to move up, down or to its neighbor fields.
AntWorld management applies the expansion rules if there is an ant on the outmost ring. Moreover, food is created on the newly created outer ring and pheromone trails are decayed and removed.
After all ants have moved and the world has been expanded possibly, the ants can still pick up or drop food. In case food is dropped on the hill, new baby ants are created.

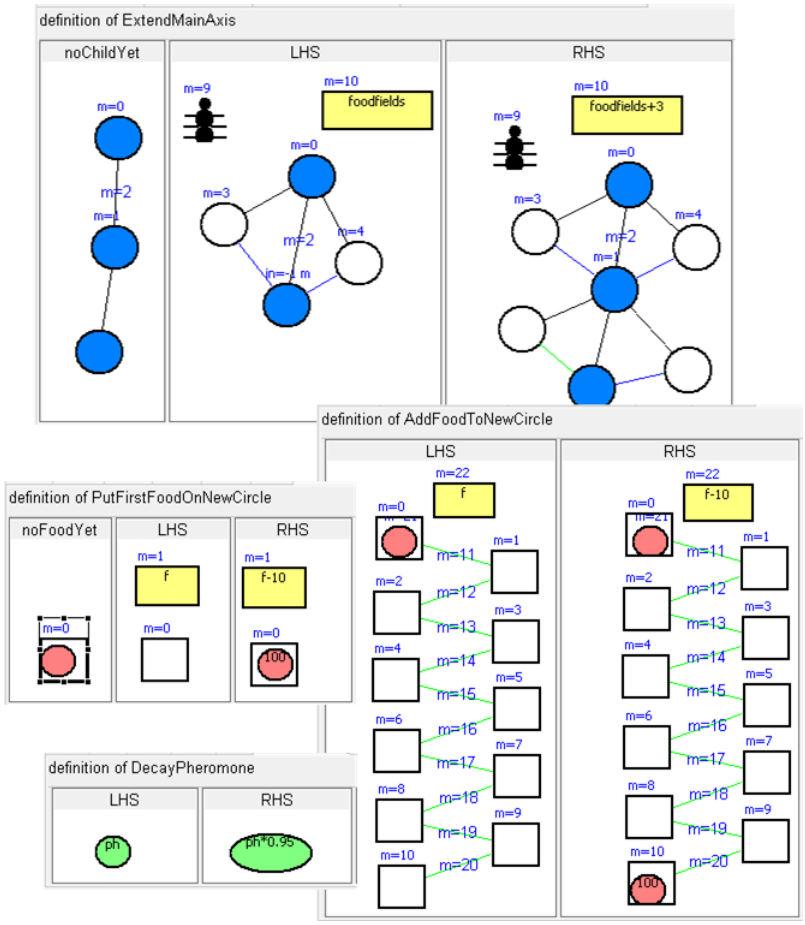

Fig. 4. Rules for AntWorld Management

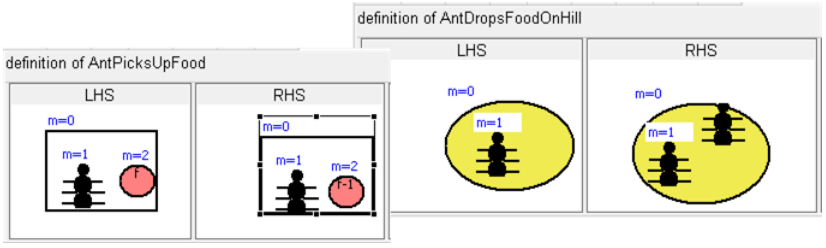

Fig. 5. Rules for Food Transport

6 Generated AntWorld Simulator

Fig. 6 shows the generated "editor", where the palette contains the steps to build and simulate the AntWorld. An entry for doing one round leads to the execution

of the complete control structure described in the previous section. All other entries are graph rules for stepwise simulation.

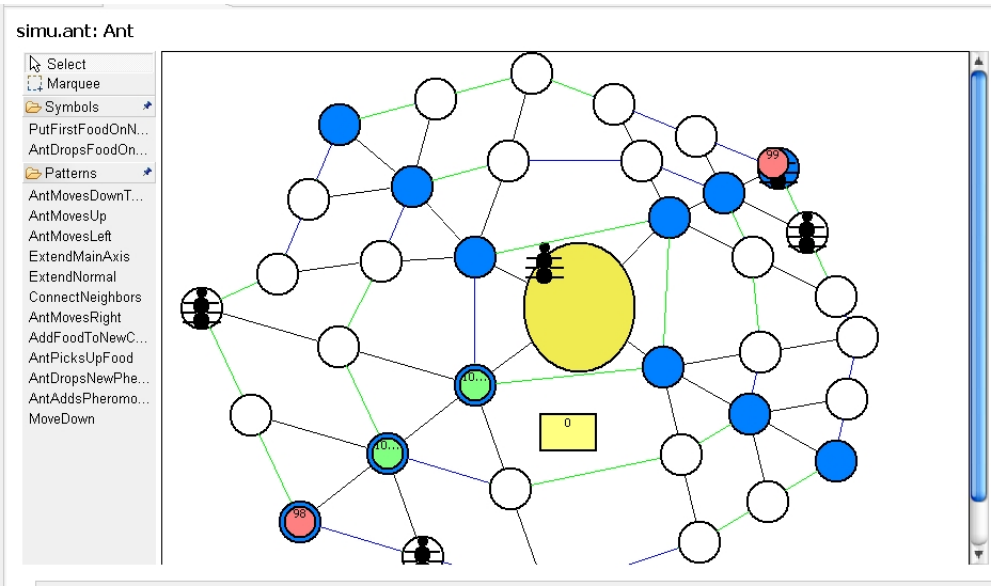

Fig. 6. Generated Environment for AntWorld Simulation

7 Conclusion

We did not run a benchmark on the Tiger simulation case study. It is not possible to run Tiger without the visual interface. If all visual GEF commands and actions would be removed, the simulation would become a pure AGG benchmark, but of course, the visual editor framework adds to the Tiger simulation runtime. The focus of Tiger is the visualization power and not the graph transformation speed. We found two things desperately missing in the current Tiger version: there are no attribute conditions supported (they had to be inserted afterwards in the generated AGG grammar), and there is no way to define control structures. For future case studies, we plan to add both features to the designer, extending Tiger from a pure editor generator to a generator of visual simulation tools.

References

1. Eclipse Consortium. *Eclipse Graphical Editing Framework (GEF) – Version 3.2*, 2006. http://www.eclipse.org/gef.
2. G. Taentzer. AGG: A Graph Transformation Environment for Modeling and Validation of Software. In Proc. AGTIVE'03, Vol. 3062 of LNCS, pages 446 – 456. Springer, 2004.