Generating the Sierpinski Triangles Using Viatra

Ákos Horváth, Máté Kovács

Department of Measurement and Information Systems
Budapest University of Technology and Economics
[ahorvath,kovmate]@mit.bme.hu

1 The Viatra2 Model Transformation Framework

The objective of the VIATRA2 (VIsual Automated model TRAnsformations [1]) is a general-purpose model transformation engineering framework that aims at supporting the entire life-cycle, i.e. the specification, design, execution, validation and maintenance of transformations within and between various modeling languages and domains in the MDA.

Tool history. The tool is being developed at the Fault Tolerant Systems Research Group at the Budapest University of Technology and Economics. Our research on model transformations of UML models is dated back to the European ESPRIT project HIDE [2]. The first versions of the tool [3] (written in Prolog) were developed between 2000 and 2003 and primarily focused on designing automated transformations of UML models into various mathematical domains of model analysis tools. Based on these initial experiments, VIATRA2 has been reengineered from scratch since 2004 (following the guidelines of a PhD thesis [4]) to better meet the requirements of MDA transformations. VIATRA2 is now written in Java and fully integrated into the Eclipse framework and accepts models of several off the- shelf industrial modeling tools. Currently, VIATRA2 is available as part of the Eclipse GMT Subproject.

Model description VIATRA2 uses the VPM metamodeling approach [5] for describing modeling languages and models. The main reason for selecting VPM instead of a MOF-based metamodeling approach is that VPM supports arbitrary metalevels in the model space. As a direct consequence, models taken from conceptually different domains (and/or technological spaces) can be easily integrated into the VPM model space. The flexibility of VPM is demonstrated by a large number of already existing model importers accepting the models of different BPM formalisms, UML models of various tools, XSD descriptions, and EMF models.

Transformation description. Specification of model transformations in VIATRA2 combines visual rule and patternbased paradigm of graph transformation (GT) [6] and the very general, high-level formal paradigm of abstract state machines (ASM) [7] into a single framework for capturing transformations within and between modeling languages.

– Queries on models are intuitively captured by graph patterns (that may have negative conditions and interact with other patterns in turn).
– Elementary model manipulations specified by graph transformation rules can also reuse predefined graph patterns in their left-hand side and right-hand side graphs.
– Then complex transformation programs are assembled by using abstract state machine constructs that provide higher-level control structures for elementary manipulation steps. Both ASM and GT rules are allowed to have input and output parameters to support information hiding.
– Code generation is treated as ordinary model-to-code model transformations, and it is supported by intelligent (model-driven) print ASM rules, and a code formatter mechanism to split the generated code into different source files.

A unique feature of VIATRA2 is the support of generic and meta-transformations [8] that allow type parameters and manipulate transformations as ordinary models, respectively. Initial experiments have been carried out to gain more practical experience in this novel field. Transformations are primarily executed within the framework by using the VIATRA2 interpreter, which uses constraint satisfaction techniques for matching graph patterns (which is typically the most expensive step of a model interpreter).

Currently, VIATRA2 provides a textual language (called VTCL [9]) for describing transformations and a separate language (called VTML) for representing models, and metamodels, and a graphical user interface for viewing and editing the VPM model space including the creation and deletion of various model elements (and traditional undo and copy operations). This GUI provides a tree view of the model space. Rich textual editors support the editing of VTCL transformations, and VTML models and meta models.

2 The Case Study

In order to demonstrate VIATRA2 we have chosen the Sierpinski triangle case study variant. The task of the case study is to simulate the creation of a Sierpinski graph, which is created by the following algorithm[1]:

1. Start with any triangle in a plane (any closed, bounded region in the plane will actually work). The canonical Sierpinski triangle uses an equilateral triangle with a base parallel to the horizontal axis.
2. Shrink the triangle by 1 2 , make two copies, and position the three shrunken triangles so that each triangle touches the two other triangles at a corner.
3. Repeat step 2 with each of the smaller triangles.

For the specification of the triangles as graphs we have followed the instruction described in [10]. The detailed description of the metamodel and transformations defined for the generation is described in 3.

[1] Taken directly from [10].

3 Solution 1

3.1 Metamodel and Pattern Matching

In a generation step first the triangles have to be collected that can be manipulated by the algorithm. Once the triangles are collected, three new nodes have to be instantiated and connected to the rest of the model. The new nodes with the new connections substitute the old edges so they have to be deleted.

From the programmers point of view the most difficult part of implementing the Sierpinski triangle generator is to create the correct triangle finder mechanisms. In our solution we adhere to the typing scheme found in the problem descrpitpion. Our solution takes full advantage of the inheritence and subtype modeling capabilities of the VIATRA2 framework.

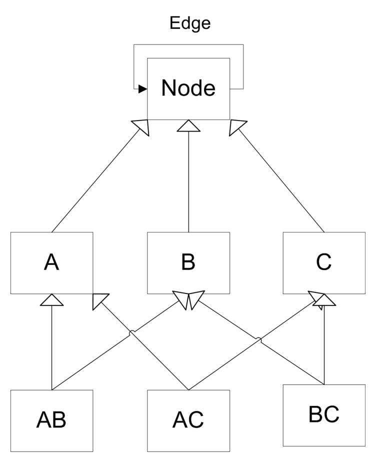

Fig. 1. The metamodel of the Sierpinski triangles

Listing 1.1. Metamodel of the Sierpinski triangles in VTML

```
entity(node);
entity(a);
entity(b);
entity(c);
entity(ac);
entity(ab);
entity(bc);

relation(e,node,node);

supertypeOf(node,a);
supertypeOf(node,b);
supertypeOf(node,c);
```

```
supertypeOf(a,ab);
supertypeOf(a,ac);
supertypeOf(b,ab);
supertypeOf(b,bc);
supertypeOf(c,ac);
supertypeOf(c,bc);
```

Fig. 1 illustrates the metamodel of the solution. The metamodel in the Viatra Textual Metamodeling Language (VTML) can be seen in Listing 1.1.

In order to have VIATRA2 match the correct triangles the type hierarchy had to be established, and directed edges had to be used. The metamodel allows us to create a very simple and elegant pattern to find the triangles for the next step.

Listing 1.2. Pattern for finding triangles in VTCL

```
pattern triangle(A,B,C,EAB,EBC,ECA) = {
a(A);
b(B);
c(C);
node.e(EAB,A,B);
node.e(EBC,B,C);
node.e(ECA,C,A);
}
```

Listing 1.2 shows the pattern in the Viatra Textual Command Language (VTCL). Capital letters stand for variables, normal letters denote direct references to modelspace elements. For instance the expression `a(A)` declares that the variable `A` is of type `a`. The declaration of edges are somewhat more complicated. The expression `node.e(ECA,C,A)` means that the variable `ECA` refers to an edge of type `node.e` that points from the entity in `C` to `A`. The pattern in Listing 1.2 matches a triangle the vertices of which are of type `a,b,c` in this order. The order is granted by the direction of the arrows.

3.2 Control Flow

The Viatra Textual Command Language (VTCL) has bot declarative and imperative sublanguages. The control flow of graph transformations or graph rewritings in this case is usually implemented using imperative concepts, graph patterns are defined in a declarative way.

Listing 1.3. The control flow of the triangle generation

```
rule generate(in MaxSteps) =
 let StepCount = 0 in seq {
  iterate if(StepCount<MaxSteps) seq {
   forall A below models("model"),
          B below models("model"),
          C below models("model"),
EAB, EBC, ECA with find triangle(A,B,C,EAB,EBC,ECA) do seq {
```

```
    //The old edges are deleted:
    delete(EAB);
    delete(EBC);
    delete(ECA);

    //The new nodes are created:
    new(ab(AB) in models("model"));
    new(bc(BC) in models("model"));
    new(ac(AC) in models("model"));

    //The new edges are created:
    new(node.e(EAAB,A,AB));
    new(node.e(EABAC,AB,AC));
    new(node.e(EACA,AC,A));

    new(node.e(EABB,AB,B));
    new(node.e(EBBC,B,BC));
    new(node.e(EBCAB,BC,AB));

    new(node.e(EACBC,AC,BC));
    new(node.e(EBCC,BC,C));
    new(node.e(ECAC,C,AC));
   }
   update StepCount = StepCount+1;
  }
  else fail;
  print("generate ended\n");
}
```

Listing 1.3 lists the main control flow of the generator program. First the triangles to manipulate are searched. The body of the `forall` keyword is run for all the matchings of the pattern called `triangle`. Inside the body of the `forall` the variables `A,B,C` are bound to graph nodes of types `a,b,c` in order. The edges are stored in the variables `EAB,EBC,ECA`. The control flow is very simple. First the old edges are deleted. In the next step the new nodes are instantiated, and finally, he new edges are connected.

The only parameter of the subroutine is the maximum step count of the generation. The outer iteration makes sure that the Sierpinski generation step is not executed more than desired. This the algorithm, and the capabilities of the transformation engine can be tested.

3.3 Performance

Using our beta staged VIATRA2 Release 3 engine, and Intel T2400@1830 MHz with 2 GByte memory, we could handle level 10 and had a JVM out of memory exception by the JVM on Level 11. That is about a quarter of a million model elements with interpreted execution and in memory model persistance. As for the run time performance due to some randomization in the matching process we measured huge differences between different executions, but overall the generation took from a couple of minutes to more then 10 minutes for level 10. Detailed performance analysis of the new Release 3 engine with the Sierpinski case study is an ongoing research.

3.4 Conclusion

Our solution was planned to be as close to the one suggested by [10] as possible. The only difference is that instead of using GT rules for the triangle generations for better performance we are using graph patterns to match the corresponding model parts and then performing the model manipulation by built in ASM rules.

The case study turned out to be useful when running profilers on the Release 3 code basis and figured out relevant bottle necks in the implementations.

References

1. Framework, V.: An eclipse gmt subproject (http://www.eclipse.org/gmt/)
2. A. Bondavalli, M. Dal Cin, D. Latella, I. Majzik, A. Pataricza, G. Savoia: Dependability analysis in the early phases of uml based system design. International Journal of Computer Systems - Science and Engineering **16**(5) (2001) 265–275
3. Csertán, G., Huszerl, G., Majzik, I., Pap, Z., Pataricza, A., Varró, D.: VIATRA: Visual automated transformations for formal verification and validation of UML models. In Richardson, J., Emmerich, W., Wile, D., eds.: Proc. ASE 2002: 17th IEEE International Conference on Automated Software Engineering, Edinburgh, UK, IEEE Press (September 23–27 2002) 267–270
4. Varró, D.: Automated Model Transformations for the Analysis of IT Systems. PhD thesis, Budapest University of Technology and Economics, Department of Measurement and Information Systems (May 2004)
5. Varró, D., Pataricza, A.: VPM: A visual, precise and multilevel metamodeling framework for describing mathematical domains and UML. Journal of Software and Systems Modeling **2**(3) (October 2003) 187–210
6. Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G., eds.: Handbook on Graph Grammars and Computing by Graph Transformation. Volume 2: Applications, Languages and Tools. World Scientific (1999)
7. Börger, E., Stärk, R.: Abstract State Machines. A method for High-Level System Design and Analysis. Springer-Verlag (2003)
8. Varró, D., Pataricza, A.: Generic and meta-transformations for model transformation engineering. In Baar, T., Strohmeier, A., Moreira, A., Mellor, S., eds.: Proc. UML 2004: 7th International Conference on the Unified Modeling Language. Volume 3273 of LNCS., Lisbon, Portugal, Springer (October 10–15 2004) 290–304
9. Balogh, A., Varró, D.: Advanced model transformation language constructs in the VIATRA2 framework. In: ACM Symposium on Applied Computing — Model Transformation Track (SAC 2006), Dijon, France, ACM Press (April 2006) 1280–1287
10. Gei, R., Mallon, C., Kroll, M.: Sierpinski triangle for the agtive 2007 tool contest (July 15 2007)