UML Models to CSP Models in MOMENT2-GT

Artur Boronat and Reiko Heckel

Computer Science Deparment, University of Leicester, UK.
ab373@le.ac.uk reiko@mcs.le.ac.uk

1 Introduction

MOMENT2 is an algebraic model management framework that permits manipulating models in the Eclipse Modeling Framework (EMF) [1]. Our goal consists in using OMG standards, such as Meta-Object Facility (MOF) [2], Object Constraint Language [3] and Query/View/Transformation (QVT) [4], as a clean interface between formal methods and model-based industrial tools that permits taking the best benefit from both at the lowest cost. In particular, we deal with Rewriting Logic (RL for short) [5] and Graph Transformations [6] on the one hand, and with Model-Driven Technology on the other hand.

MOMENT2-GT constitutes an extension of MOMENT2 that provides support for graph-based transformations. The syntax for defining graph patterns is directly based on the QVT Relations language. We present the solution of a case study of the AGTIVE contest by using it: the transformation of UML Activity models into CSP models.

The paper is structured as follows: Section 2 introduces the main features of MOMENT2-GT; Section 3 describes the solution to AGTIVE case study 2; Section 4 provides a brief discussion of the experience; and Section 5 finally summarizes the main outcomes.

2 MOMENT2-GT Features

MOMENT2 constitutes an efficient framework that enhances the experimentation of practical graph-based model transformation language features: MOF-based concepts, MOF introspection and structural reflection, graph pattern matching, negative application conditions, conditional production rules, OCL support, textual concrete syntax of transformation definition languages; and formal verification techniques: checking invariants through reachability analysis, model checking, and theorem proving among others. Indeed, MOMENT2 constitutes an abstraction on top of RL that permits dealing with MOF-based models and their consistent manipulation (keeping the graph structure, for example). The framework is entirely written in Maude [7], an efficient term rewriting system that provides support for RL. MOMENT2 has been designed in order to take advantage of Maude's formal analysis techniques. MOMENT2-GT reuses its core functionality for graph transformation purposes:

- MOMENT2-GT provides a textual QVT-based language for defining production rules, where graph patterns are defined as in the QVT Relations language [4] and attribute values can be queried and manipulated by means of OCL expressions.
- Support for textual concrete syntax. The grammar of the transformation language itself is defined as an algebraic signature. A transformation definition is a term, which is compiled into a Maude module. Therefore, our framework can be extended with new concepts and syntax very easily.
- Definition of production rules either as equations, to specify functional behavior, or as rewrite rules, to specify non-deterministic and possibly concurrent behavior. A MOMENT2-GT transformation definition can be viewed as a rewrite theory.
- EMF is reused as a metamodeling front-end so that graphical editors and persistence can be customized easily. At the time of writing this paper, the EMF concepts that are supported are: class definition with attributes and references; abstract classes; class specialization; multiplicity features on references (but only one-bounded attributes), including ordering and uniqueness; opposite references; and containment references. Since MOMENT2-GT is a self-contained framework in Maude, adopting other metamodeling front-ends such as Microsoft DSL Tools [8] is straightforward.
- In MOMENT2-GT, edges are updated automatically when they are defined as opposite references in the metamodel. On the one hand, if one of two opposite references is removed, the other one is also removed. On the other hand, if one of them is updated, the other one is also updated. In addition, an object in an EMF model can be the root of a subtree (taking into account containment references). When an object of this kind is deleted, all objects in the subtree are deleted and dangling edges are removed as well. These actions are carried out automatically by MOMENT2-GT in what we call *consistency checking*. MOMENT2-GT permits disabling it when there are no dangling edges. We will study the performance of the UML2CSP transformation by taking into account both possibilities.

3 AGTIVE Case study 2: UML To CSP Transformation

In this Section, we provide a summary of two solutions for the UML2CSP case study. The first transformation can produce dangling edges, and MOMENT2-GT takes care of the consistency of the resulting graph. The second transformation is designed to avoid such dangling edges so that MOMENT2-GT will be used without consistency checking.

The activity metamodel that we have taken into is in Appendix A and the CSP metamodel is in Appendix B. We have introduced a slight modification in the CSP metamodel of the case study. We have introduced a class called *ProcessIdentifier* that is related to the class *ProcessAssignment* by means of a containment reference *processIdentifier*. The *Process* class has a reference to the *ProcessIdentifier* class. This fact prevents a *ProcessIdentifier* instance from being contained within more than one *ProcessAssignment* instance.

In our solution, we process the objects that constitute the input activity model generating objects in the resulting CSP model. The idea behind the transformation definition is to delete activity nodes whenever they have been processed. Our two solutions are roughly the same. We will explain then the common parts and we will indicate what the differences are.

3.1 UML To CSP Transformation Definition

The transformation definition has 14 production rules that can be grouped as follows:

- *CspContainer*: there is an initial rule that generates the root node of the CSP model. In addition, a *ProcessIdentifier* node with name *SKIP* is created in the LHS of a *ProcessAssignment* object.
- *ProcessIdentifier*: one rule is used to generate a *ProcessAssignment* for each *ActivityEdge* node. A *ProcessIdentifier* node is also generated in the LHS of the process assignment.
- *FinalNode processing*: one rule completes a process assignment with a *ProcessIdentifier* node with name *SKIP* for each *ActivityEdge* that points to a *FinalNode* node in the input model.
- *Event*: One rule generates an event process (*Event* node) for each *Action* node in the activity model. This event process is contained in a process assignment, where its LHS is the process that corresponds to the input edge of the *Action* node in the activity model, and the target process of the event process is the process that corresponds to the output edge of the *Action* node in the activity model. This rule is defined in Appendix C (Fig. 3) by using the MOMENT2-GT language, where graph patterns are defined in a production rule by using the syntax of the QVT Relations language. The same production rule has been defined in Tiger [9] in Appendix C (Fig. 4) as well. This permits comparing the textual character of the MOMENT2-GT language with the graphical character of Tiger.
- *MergeNode*: One rule processes a *MergeNode* node in the input activity model. In this version of the transformation definition, we can only take into account *MergeNode* nodes with two input edges.
- *DecisionNode*: Three rules are used to process *DecisionNode* nodes in the input activity model. Two are used to define the conditions in the RHS of the corresponding *ProcessAssignment* node. The other one is used to delete the *DecisionNode* node.
- *ForkNode* : Three rules are used to process *ForkNode* nodes in the input activity model. Two are used for defining *Concurrency* nodes in the RHS of the corresponding assignment corresponding. The third one is used to delete the *ForkNode* object in the input model when it has been already processed.
- *JoinNode* : Two rules are used to process *JoinNode* nodes in the input activity model. One assigns the process that corresponds to the output edge of the *JoinNode* node to the *process* that corresponds to one of its input edges. The other one assigns the *SKIP* process to the process that corresponds to each one of the rest of input edges.

– *ActivityCollection*. The last rule deletes the container node that constitutes the root object of the activity model. Since MOMENT2-GT deletes trees of objects, by deleting the root object of the model, we delete the entire activity model.

3.2 Avoiding Consistency Checking

In MOMENT2-GT, we can speed up the transformation execution process by avoiding consistency checking. This should be only used when no dangling edges are produced. To avoid them, we have to make explicit all the edges that should be removed or updated in a production rule. For example, in the rule of Appendix C, we have to add the *container* edge in the *Action* node pattern of the LHS of the production rule as follows:

```
'a : 'Action {
    'container = 'col : 'ActivityCollection {
        'activities = 'a : 'Action { }
    },
    'name = 'aName:String
}
```

In the RHS of the production rule, we have to indicate that the *container* and the *activities* edges are removed. The *Action* node will be deleted as well.

```
'col : 'ActivityCollection { }
```

Another difference is that the rule that deletes the activity model at the end of the transformation should delete all objects in the input model. The reason is that we do not want to delete trees. Therefore, we have to add one more rule to delete all *ActivityEdge* nodes.

4 Discussion

We provide an average of the time measurements that have been obtained during 10 experiments. The experiments have been performed on a Core DUO 2Ghz with 2Gb RAM, using Ubuntu 7.04. The transformation that can produce dangling edges was performed in an average time of 1431.2 ms by Maude. The transformation that was designed to avoid dangling edges was performed in an average time of 885.4 ms by Maude (by disabling the aforementioned consistency checking).

MOMENT2-GT provides support for consistent hierarchical graph manipulation (MOF-based model manipulation) by following the *Single Pushout* approach. It relies on RL, raising the level of abstraction from flat terms to graph structures. Indeed, the MOMENT2-GT model transformation language can be viewed as syntactic sugar for defining specific RL theories. Thus, a MOMENT2-GT model transformation definition can be viewed as a rewrite theory in the end.

Therefore, we can apply RL-based formal analysis techniques to model-based systems in a straightforward way. In particular, this formal reasoning can be supported by the various formal tools that are available in the Maude environment, including an inductive theorem prover, a model checker, and tools for checking sufficient completeness, confluence, and termination of specifications.

A disadvantage in our approach is that it lacks of graphical concrete syntax. Comparing a production rule in Tiger and in MOMENT2-GT in Appendix C shows, at a first glance, that our approach is not the most appropriate for communication purposes. However, for expert users, a textual-based syntax may offer editing facilities that are difficult to achieve in a graphical approach: copy & paste, text replacement, etc. In addition, MOMENT2-GT constitutes a framework that is defined at a high level of abstraction in Maude. Therefore it is ideal for experimenting with new model transformation features, keeping in mind a realistic approach in terms of efficiency. Since MOMENT2 already provides support for EMF, reusing Tiger as front-end for MOMENT2-GT would be quite straightforward.

5 Conclusions

We have provided a solution for the UML2CSP case study of the AGTIVE contest by using MOMENT2-GT. MOMENT2-GT constitutes an efficient graph-rewriting engine that is powered by Maude, that is integrated into the EMF and that follows OMG standards: MOF, OCL and QVT. MOMENT2-GT transformation language is directly based on RL, raising the level of abstraction from flat terms to graph structures. This fact permits using formal analysis techniques in graph-based transformations and model-driven development processes.

Acknowledgments. This work has been partially sponsored by the Spanish Government under the National Program for Research, Development and Innovation, DYNAMICA Project TIC 2003-07804-C05-01 and META Project TIN2006-15175-C05, and by the project SENSORIA, IST-2005-016004.

We are grateful to prof. José Meseguer for his help, advice and corrections in the development of the MOMENT2 kernel.

References

1. Eclipse Organization: The eclipse modeling framework (2007) `http://www.eclipse.org/emf/`.
2. Object Management Group: Meta Object Facility (MOF) 2.0 Core Specification (ptc/06-01-01) (2006) `http://www.omg.org/cgi-bin/doc?formal/2006-01-01`.
3. Object Management Group: OCL 2.0 Specification (2006) `http://www.omg.org/cgi-bin/doc?formal/2006-05-01`.
4. Object Management Group: MOF 2.0 QVT final adopted specification (ptc/05-11-01) (2005) `http://www.omg.org/cgi-bin/doc?ptc/2005-11-01`.

5. Meseguer, J.: Conditional rewriting logic as a unified model of concurrency. Theoretical Computer Science **96**(1) (1992) 73–155
6. Rozenberg, G., ed.: The Handbook of Graph Grammars, volume 3. World Scientific (1999)
7. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: All About Maude: a High-Performance Logical Framework. Springer (2007)
8. Microsoft Corp.: Domain-specific language tools (2007) `http://msdn2.microsoft.com/en-us/vstudio/aa718368.aspx`.
9. TU Berlin, TFS: Tiger EMF Transformation Project (2007) `http://tfs.cs.tu-berlin.de/emftrans/`.

A Activity Metamodel

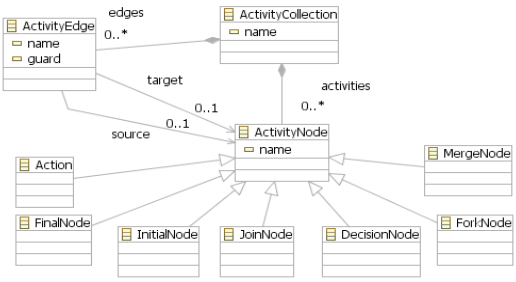

Fig. 1. Activity Metamodel.

B CSP Metamodel

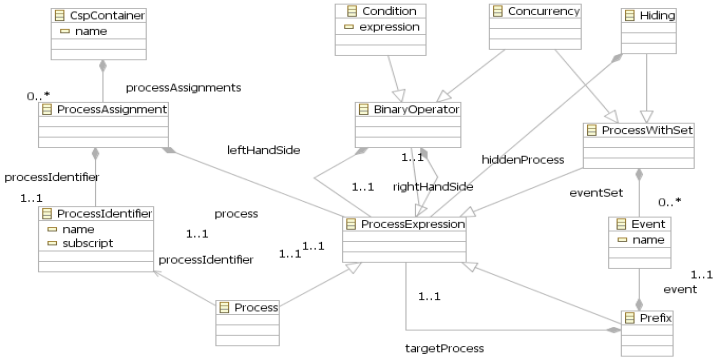

Fig. 2. CSP Metamodel.

C Production Rule in MOMENT2-GT and in Tiger

```
eq 'rule4-Action-->Event
{
    lhs {
        '1 : 'ActivityEdge {
            'name = 'inEdgeName:String,
            'target = 'a : 'Action {
                'name = 'aName:String
            }
        }
        '2 : 'ActivityEdge {
            'name = 'outEdgeName:String,
            'source = 'a : 'Action {}
        }
        '3 : 'ProcessAssignment {
            'processIdentifier = '4 : 'ProcessIdentifier {
                'name = 'inEdgeName:String
            }
        }
        '5 : 'ProcessIdentifier {
            'name = 'outEdgeName:String
        }
    };
    rhs {
        '1 : 'ActivityEdge {
            'name = 'inEdgeName:String
        }
        '2 : 'ActivityEdge {
            'name = 'outEdgeName:String
        }
        '3 : 'ProcessAssignment {
            'processIdentifier = '4 : 'ProcessIdentifier {
                'name = 'inEdgeName:String
            },
            'process = 'inP : 'Prefix {
                'targetProcess = 'inP-targetProcess : 'Process {
                    'processIdentifier = '5 : 'ProcessIdentifier {
                        'name = 'outEdgeName:String
                    }
                },
                'event = 'inP-event : 'Event {
                    'name = 'aName:String
                }
            }
        }
    };
}
```

Fig. 3. Action To Event production rule in MOMENT2-GT.

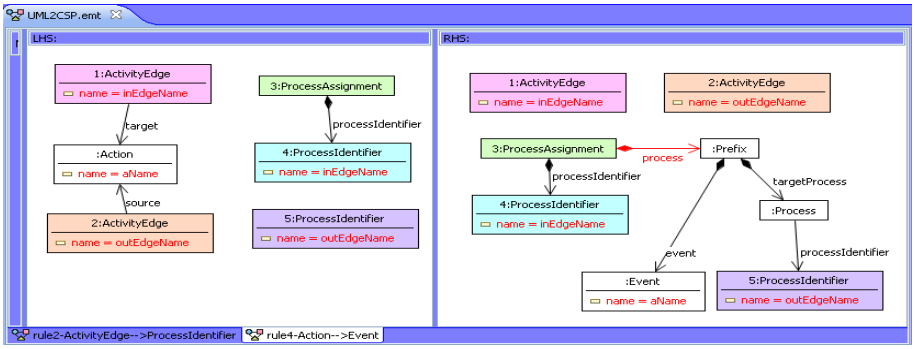

Fig. 4. Graphical representation of a production rule in Tiger.