AUTOMATED PROGRAM GENERATION IN VIATRA

Dániel VARRÓ
Advisor: András PATARICZA

I. Introduction

Although the Unified Modeling Language (UML) has become the de facto standard visual modeling language of object–oriented design, both academic investigations and engineering experiments have revealed several shortcomings regarding, especially, its imprecise semantics and the lack of flexibility in domain specific applications [1]. Recently, the UML 2.0 Request For Proposal issued by the Object Management Group has addressed to re-architecture the single and monolith language into a family of languages with individually defined semantics based on a kernel metamodeling language.

However, as the formal semantics of different views of the system (i.e., separate diagrams like class diagrams, statecharts, sequence diagrams, etc.) might be defined in different semantic notations (e.g., by Petri nets, SOS rules, graph transformation systems etc.), the integration of such local views into a consistent global view of the system requires a precise specification of transformations *within* and *between* UML models.

VIATRA (VIsual Automated model TRAnsformations [2]; see Fig. 1) is a prototype tool that provides a general and automated framework for specifying transformations between arbitrary models conforming to their MOF metamodel [3]. From visual model transformation rules defined in a UML profile, VIATRA automatically generates a Prolog program for the implementation. The precise theoretical background of the transformations is formalized by means of graph transformation systems [4].

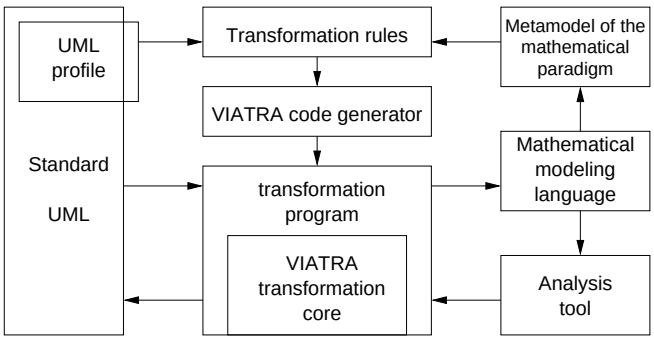

Figure 1: The architecture of VIATRA

II. Automated Program Generation in VIATRA

The user interface of VIATRA has a uniform representation of metamodels, models and their transformations in the form of UML models exported in their standard XMI format by a commercial CASE tool. The source and target languages of a model transformation are defined by a corresponding **metamodel**, which provides a uniform description for both UML and mathematical languages (such as Petri nets, state transition systems, etc.). The elementary model transformation steps are defined by specially structured **graph transformation rules** [2] while the entire computation is constrained by a **control flow graph**.

To demonstrate that VIATRA is a general purpose transformation engine, the *implementation of model transformation systems is specified by model transformation systems,* which is conceptually similar to the well–known bootstraping techniques of compiler design. Since there is a huge abstraction gap between a visual UML-based specification of the transformation and even such a high-level programming language as Prolog the entire program generation is split into several subtransformations, which decreases the complexity of the individual steps.

1. Model transition systems are specified in a UML CASE tool (Rational Rose was used for our experiments), and exported in the standard XMI format.

2. This *UML model* is transformed into a *GraTra model*, which conforms to the metamodel of *graph transformation systems*.

3. In VIATRA, model graphs are represented as predicates in a fact database. For this reason, the previous *GraTra model* is projected into first into a sequence of logic terms for each rule (called *Logic model*).

4. The bridge between the visual (graph based) and the textual language of Prolog is provided by the parse tree of the Prolog code. In this sense, the *Logic model* is transformed into a corresponding *Prolog parse tree*, and the *target Prolog code* is printed out by traversing this tree in an in-order way.

The automated program generation of VIATRA allows the transformation designers to focus on the *design* of a model transformation rather than on the *implementation*. The quality of an automatically generated executable code is much higher than a manually written target program. Moreover, once the automated program generator is completed, the time and workload related to the design of a single transformation is drastically decreased.

III. Experimental Results

VIATRA has already been applied successfully to provide an automated implementation of transformations specified by means of model transformation systems. An industrial strength transformation is the automated implementation of [5] where a mapping is presented from UML Statecharts to Extended Hierarchical Automata (EHA) that provides formal operational semantics for statecharts. In [6], we formalized the entire transformation by model transformation systems with more than 40 rules. The entire code generation step required 73 seconds processing three large XMI files (3 MB each with approximately a hundred thousand graph objects altogether). Afterwards, a concrete SC-EHA model transformation of a statechart with 20 states and 15 transitions terminated within a second.

The completeness of UML statechart specifications in a dependable environment has been investigated in [7]. Recently, an automated verification program has also been implemented based on VIATRA.

References

[1] Cris Kobryn. UML 2001: A standardization Odyssey. *Communications of the ACM*, 42(10), 1999.

[2] Dániel Varró, Gergely Varró, and András Pataricza. Designing the automatic transformation of visual languages. *Science of Computer Programming*, December 2001. In print.

[3] Object Management Group. *Meta Object Facility Version 1.3*, September 1999.

[4] Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations*, volume 1: Foundations. World Scientific, 1997.

[5] D. Latella, I. Majzik, and M. Massink. Automatic verification of UML statechart diagrams using the SPIN model-checker. *Formal Aspects of Computing*, 11(6):637–664, 1999.

[6] Dániel Varró and András Pataricza. Mathematical model transformations for system verification. Technical report, Dept. of Measurement and Information Systems, Budapest University of Technology and Economics, May 2001.

[7] Zs. Pap, I. Majzik, and A. Pataricza. Checking general safety criteria on UML statecharts. In *Computer Safety, Reliability and Security (Proc. 20th Int. Conference, SAFECOMP-2001)*.