Journal of Visual Languages and Computing 20 (2009) 252–268

Contents lists available at ScienceDirect

Journal of Visual Languages and Computing

journal homepage: www.elsevier.com/locate/jvlc

Generic and reflective graph transformations for checking and enforcement of modeling guidelines

Elodie Legros *, Carsten Amelunxen, Felix Klar, Andy Schürr

Real-Time Systems Lab, Technische Universität Darmstadt, Germany

ARTICLE INFO

Keywords:
Graph transformation
Generic rules
Reflection

ABSTRACT

In the automotive industry, the model driven development of software, today considered as the standard paradigm, is generally based on the use of the tool MATLAB Simulink/Stateflow. To increase the quality, the reliability, and the efficiency of the models and the generated code, checking and elimination of detected guideline violations defined in huge catalogs has become an essential task in the development process. It represents such a tremendous amount of boring work that it must necessarily be automated. In the past we have shown that graph transformation tools like Fujaba/MOFLON allow for the specification of single modeling guidelines on a very high level of abstraction and that guideline checking tools can be generated from these specifications easily. Unfortunately, graph transformation languages do not offer appropriate concepts for reuse of specification fragments—a MUST, when we deal with hundreds of guidelines. As a consequence we present an extension of MOFLON that supports the definition of generic rewrite rules and combines them with the reflective programming mechanisms of Java and the model repository interface standard Java Metadata Interface (JMI).

1. Introduction

Nowadays, model-driven development is common practice within a wide range of automotive embedded software development projects. In this domain, the standard modeling language UML does not meet the requirements of the developers and, therefore, is neglected in favor of the MathWorks Matlab Simulink/Stateflow (Matlab SL/SF) [14] environment which is better adapted for specifying, designing, implementing, and checking the functionality of new control functions. In fact, Simulink supports a block-oriented style of modeling that combines the dataflow programming paradigm with differential equation solvers, whereas Stateflow adds a discrete event and state-oriented style of modeling.

To improve the correctness and the efficiency of models and prevent typical modeling problems, generally accepted modeling guidelines such as the MathWorks Automotive Advisory Board catalog [13] are usually adopted. These modeling guidelines are either manually or automatically checked during audits using tools like the MathWorks Model Advisor [14]. Nevertheless, the modeling guidelines are numerous and, for huge models, this can add up to a few hundreds or even thousands of violations that must be corrected manually by the modeler.

The automation of such a task would be obviously of great advantage. Nevertheless, we are not aware of any tool support in this direction except of our Matlab SL/SF Model Analysis and Transformation Environment (MATE) [21].

In addition to the urgent need for such a tool, another motivation for starting the MATE project was our observation of the very low level of abstraction concerning the implementation of modeling guidelines for which imperative programming languages are generally used. Therefore, the realization of really complex checks is

* Corresponding author.
E-mail address: legros@es.tu-darmstadt.de (E. Legros).

1045-926X/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jvlc.2009.04.005

almost infeasible as well as the development of even more complex model transformations that eliminate identified guideline violations automatically. Our experience showed us that graph transformations generally offer a significantly better support for the specification and implementation of modeling guidelines and refactorings [3]. Though we are not completely satisfied with the specification of some kinds of modeling guidelines with the currently used graph transformation language story driven modeling (SDM). For instance, numerous guidelines require the definition of very similar graph transformations, which is a repetitive and time-consuming task for the developer. We are convinced that it could be significantly improved by adopting genericity and reflection concepts from standard programming languages such as Ada or Java. We introduced these concepts in a previous paper [2]. This article is an extended version of the latter. In addition to the description of the proposed features, we now present a more formal definition of them as part of a metamodel describing the SDM syntax. We also describe the techniques which are used to realize these additional graph transformation concepts. Even if the specification of modeling guidelines was the first motivation for our work, our proposal is not restricted to this area, but is applicable to any domain in which visual graph transformations can be used. Therefore, we present here an example that applies the new concepts to a refactoring operation, which is partially realized by graph transformations.

The rest of this paper is structured as follows. In the next section, we present the MATE project and then in Section 3 describe the syntax of the graph transformation language SDM and the overall structure of the Matlab Simulink metamodel, as well as the running example used in the following sections. We then present in Section 4 the Java Metadata Interface (JMI) which is used for code generation purposes and show how the low-level concept of the JMI interface for generic and reflective programming purposes can be lifted to the higher level of abstraction of the graph transformation language SDM. Section 5 describes our proposed enhancements for generic and reflective model transformations on the basis of the running examples, whereas Section 6 provides a description of the extended SDM syntax as metamodel. This section also presents the mapping between the new features and the generated JMI-compliant code, and explains how to adapt the code generation to the generic and reflective graph transformations. Section 7 presents an application of our proposal for the automation of refactoring operations followed by Section 8 which gives an overview of related approaches. Finally, Section 9 concludes this paper with our plans for future work concerning the design and implementation of a more powerful graph transformation environment.

2. The MATE project

The Matlab SL/SF Model Analysis and Transformation Environment (MATE) provides support for semi-automatic checking and enforcement of modeling guidelines as well as for design pattern instantiation, interactive model refactoring and beautifying operations. MATE is a joint project in response to the urgent need of the automotive industry for more sophisticated tool support to assist software developers using Matlab SL/SF with the maintenance and quality assurance problems of everyday life programming. Because Matlab SL/SF is used for the development of safety-critical embedded systems, model audits have become necessary steps that must be executed with rigor, but are time-consuming and thus cost-intensive processes. A solution to reduce the effort of reviewing is to ensure the quality of a model already during its development. In practice, catalogs of modeling guidelines are defined and the models are continuously and automatically checked according to these guidelines during the development.

Analysis as well as refactoring of Matlab SL/SF models requires full access to the model repository of Matlab, which is possible through an API written in m-script, a proprietary script language. Due to the fact that both the used language and the tool's API evolved over many years, learning how to program reliable model checks and transformations using this approach costs time and efforts. Furthermore, m-script is not very well suited for the specification of modeling guidelines. As shown in Amelunxen et al. [3], model checks written in m-script are generally not very easy to read or understand. MATE overcomes this problem by providing a layer of uniform API adapters on top of which visual graph queries and transformations can be developed in a more human friendly manner and on a considerably higher level of abstraction. The specification of these graph queries and transformations is realized with the help of the Fujaba graph transformation tool [17] and its metamodeling add-on MOFLON [1]. A more detailed description of the MATE system architecture and its functionality as well as its integration with the MathWorks tool suite is out-of-scope of this paper and may be found in Stürmer et al. [21].

3. Matlab metamodel and SDM

Inside MATE, guideline specifications are based on a MOF 2.0 [18] compliant metamodel of Matlab SL/SF. This metamodel acts as graph schema for the specification of graph transformation rules. The technique of graph transformations is on the one hand applied for the detection of incorrect models as well as, on the other hand, for the (semi)automatic repair of identified errors. MATE uses the visual graph transformation approach of story driven modeling [23]. In the following we give a brief overview of this approach and its syntax followed by some examples of guideline specifications.

3.1. The story driven modeling syntax

The central idea of story driven modeling is the combination of common UML activity diagrams with graph transformations. The essential graph schema is modeled as UML/MOF class diagram. An activity diagram is used to specify the behavior of exactly one operation of a schema class by specifying the control flow concerning