Describing Modeling Deltas by Model Transformation

Dilshodbek Kuryazov, Jan Jelschen, Andreas Winter
Carl von Ossietzky Universität, Oldenburg, Germany
{kuryazov,jelschen,winter}@se.uni-oldenburg.de

ABSTRACT

Since large scaled software models typically exist in many revisions, extraction and representation of differences between versions is a crucial issue of model version systems. While handling model differences is playing an important role in evolution of models, there is a need for appropriate techniques to represent model differences. This paper shows a meta-model-generic and transformation based approach to the representation of model differences. Domain specific language is generated to represent model differences. Differences are mapped to a set of model transformation rules. To demonstrate the approach, it is applied to two versions of an activity diagram.

Keywords

metamodeling, modeling delta, model transformation

1. MOTIVATION

Software models evolve over time, undergoing corrections, extensions, and updates. The evolution of software models requires collaboration of several designers. In order to provide the collaborative work of development teams and keep track of previous versions of models, model version control approaches have to provide several services, such as *calculating*, *visualizing*, *representing*, *merging*, and *analyzing* differences between consecutive versions of software models. During the collaboration, applying changes to models concurrently results in having several versions which differ from each other. This paper intends to define transformation based means to represent model differences.

Currently, most version control tools provide a text-based approach to represent differences as plain text documents (for instance, *diff*, which is applied in RCS [14], CVS [11], SCCS [1] and subversion [6]). Models can be represented as text, as well like XML Metadata Interchange (XMI) format [15]. Model differences are only viewed in a text-based way, but do not refer to modeling concepts. Information on changed modeling concepts are blurred, and can not be used for later analyses. Text-based version control systems do not provide detailed and complete techniques to represent differences suitable for model evolution. For example, changing the target of a control flow in an activity diagram is represented as changing some XMI code instead of referring to activity diagram concepts like control flows which ends at another activity.

The information obtained from the difference calculation process (e.g. [3], [9], [5]) needs to be properly represented in a difference model, usually called *modeling delta*, so that it can be used for subsequent analysis and manipulation. For this reason, appropriate techniques and tool support for the representation of version differences of software models are required.

There have been some works proposing representations of model differences. A meta-model-independent approach by Cicchetti et al. [7] provides operations that can be, under certain conditions, composed sequentially or in parallel, in order to represent more complex modifications. It introduces a set of model transformation rules in the ATLAS Transformation Language [10].

A top-down difference calculation and representation algorithm is the DSMDiff algorithm [3]. Models and metamodels are considered as graphs. The modification operations conform only to their internal format. It locks model development into a single tool restricting its exploitation in other modeling environments. Another graph based approach for differences calculation is the SiDiff algorithm [12]. In this approach, models are represented as graphs and detection of differences is based on a search algorithm.

The approach presented here is *meta-model-generic*, i.e. difference operations can be generated for arbitrary metamodels. The model modifications are carried out by means of difference models which identify sequences of modification operations such as `add`, `delete` and `change`. These operations are applicable to each of the model elements. This introduces a *transformation-based* representation of modeling deltas by transformation rules referring to a meta-model. Meta-models are a basic part of model transformations, as they allow structural definitions (i.e. abstract syntax) of the modeling language. Meta-models give a collection of concepts within a certain (domain-specific) modeling language. In this sense, applying the aforementioned basic operations to model elements results in a sequence of operations within a *domain-specific language* (DSL), which represents model differences. This allows for representing model differences in terms of the actual modeling concepts. Consequently, the operations of the DSL are represented by a set of transformation rules. Sequences of these transformation rules form an executable description of model differences.

Instead of making multiple copies of the same model artifacts, transformation-based versioning techniques only require to store an initial model and several modeling deltas. Another version of a given model will be derived by applying these rules to a given model. To express the idea, UML activity diagrams [2] are used as an example.

There are a number of model transformation languages, such as FUJABA [16], ATL [10], AGG [13], and VIATRA2 [8]. Here, the VIATRA2 (VIsual Automated model TRAns-

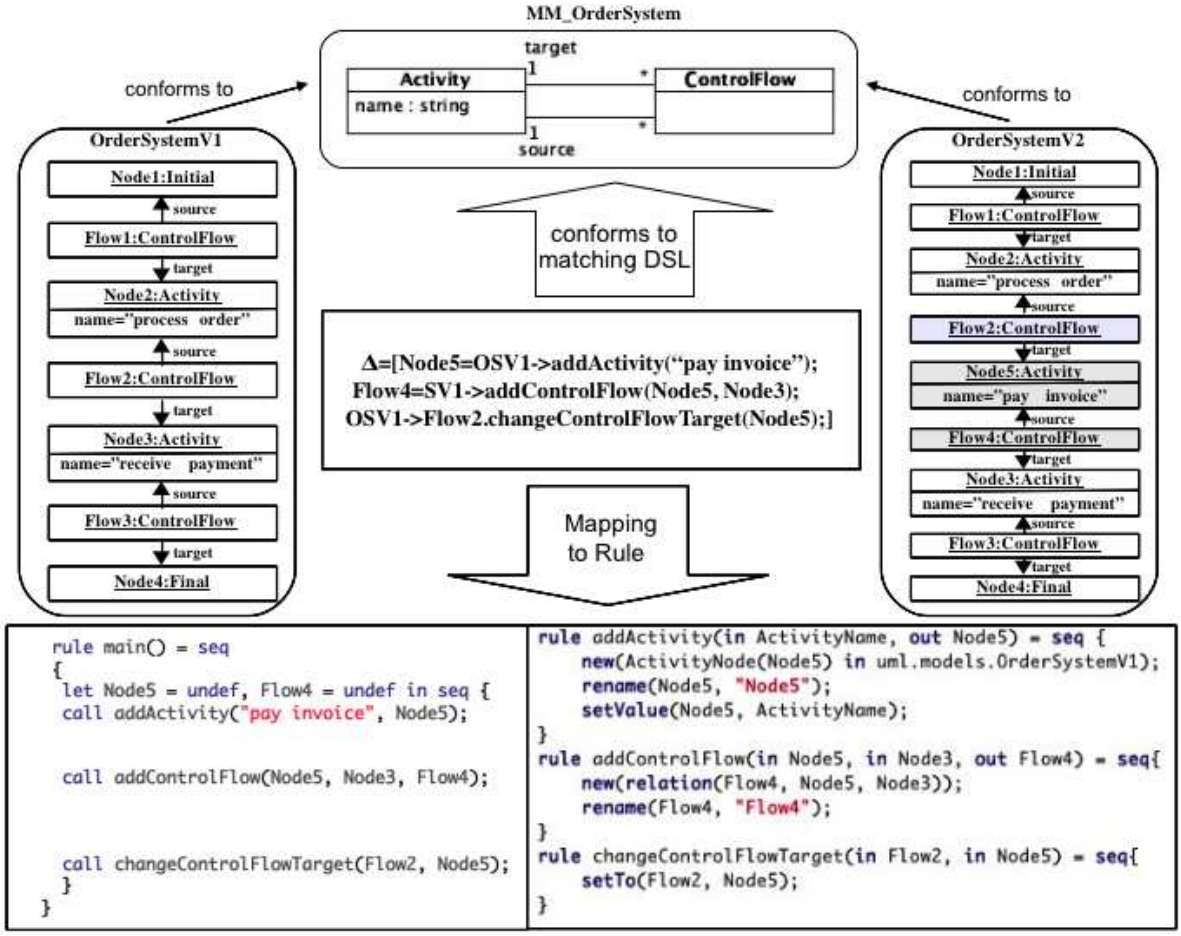

Figure 1: Example representing a simple implementation of the approach.

formation) rule- and pattern-based framework is used as concrete implementation, chosen for its general-purpose model transformations. It provides a model specification space for model manipulation, including a model editor, import-/export facilities, and visualization features. All of the other tools mentioned provide some of these features, but none of them provide all, at least not to the same extent.

The paper is structured as follows: Section 2 gives a brief overview of the approach, describing an example as motivation and illustration. Then, an agenda for future work is given in Section 3.

2. APPROACH

Graphs are defined to represent the abstract syntax of modeling languages, and applied to the representation of visual languages [4]. They are well suited to use for models to represent all kinds of artifacts. Additionally, graphs are accompanied by meta-model-based technologies to define, manipulate, analyze, and transform graphs. However, external representation of meta-models and models are defined by directed graphs in VIATRA2 framework.

Models (meta-models and instance models) consist of a set of connected elements. In the UML standard, meta-models are defined using class diagrams, that may include attributes with types, generalizations and associations. In terms of *graphs*, classes are nodes and relations between classes are edges of a graph. In the sense of instance models, the standard notation of activity models are viewed as `nodes` (action, initial, final, control, object, fork, merge, join, flow final, control flow, object flow) of the graph including attributes and associations between nodes are *edges*. In this sense, nodes and edges can be modified.

To generate the operations of the DSL, basic operations (add, delete, and change) are applied to the concepts.

In general, the following basic definitions are specified regarding the difference operations of the DSL:

- *Additions:* new nodes are added to the new version of a graph that did not exist in the old version.
- *Deletions:* existing nodes are deleted from the new version of a graph that existed in the old version.
- *Changes:* attributes of nodes are changed during the evolution from the old graph to the new one.

As an example, Figure 1 shows the simplified view of the meta-model for activity diagrams, and graph representations of two versions of an activity diagram. On the left-hand side, the initial version of the graph view of a simple activity diagram (*OrderSystemV1*) is shown. It has a start node, two activity nodes, and a final node, connected by control flows. It then evolved into the second, right-hand version (*OrderSystemV2*), in which one activity node (*"pay invoice"*) has been added *between* the two original ones, i.e. one control flow was re-targeted and a new one was added, as well.

In this example, both activity diagrams conform to the same meta-model. The operations of the DSL are specified

based on the fundamental operations. An instance of that DSL is depicted in the center, representing the differences between the two activity model versions. These difference operations conform to the meta-model, generated from the activity meta-model and the three basic operations. A meta-model describes common properties of its instances, and by applying the basic operations to the each of its concepts, the syntax of the DSL can be derived (e.g. for activity nodes: *addActivity*, *deleteActivity*, *changeActivityName*). On the bottom, the figure shows a simple set of VIATRA2 rules implementing the operations expressed using the DSL.

While the sample delta includes a simplified view of the model manipulation operations, such as `add`, `delete`, and `change`, in the same way, the VIATRA2 implementation also provides rules to add, delete, and change model elements. Consequently, the delta clearly is implemented with model transformation rules.

The set of rules have a `main` rule including *call* operators on the left side of Figure 1, and the set of rules `addActivityNode`, `addControlFlow`, `changeControlFlowTarget` on the right side corresponding each operation of the DSL.

In terms of activity models, the *change* operation exists only for attributes. The other elements of the model can only be created or deleted.

In this case, a meta-model is needed 1) as basis for defining the delta DSL and 2) as domain and range for the transformations implementing the delta operations.

In VIATRA2 framework, the model elements can be called directly by the rules to manage modifications of the model. The names of the models and meta-models need to be defined at the beginning of the model manipulation module. In order to execute model transformation rules, the concepts of UML 2 are adapted to the VIATRA2 framework and placed in the *uml.metamodels* package. Instance models themselves are created in the model editor in model space.

The transformation-based specification of the DSL improves the applicability and integrability of the approach. Moreover, only initial model versions and several applicable rules need to be stored. Other, older or newer versions can be retrieved by applying corresponding modeling deltas to an initial model.

3. OUTLOOK

This paper discussed the issue of representing model differences. A meta-model-generic transformation-based approach, allowing to retrieve model versions for graph structured models, was presented. To demonstrate the operations of the DSL, they were implemented using a model transformation language.

Applying the basic operations (*add*, *delete*, and *change*) to model elements leads to a representation of model differences. These delta operations are generated from the meta-models of modeling languages. In this case, UML activity diagrams were used as an example to illustrate the viability of the approach. In the same way, the meta-model-generic approach can be applied to other meta-models to generate appropriate domain-specific languages, as well.

As mentioned, the model version and management which intended to provide collaborative work consists of several issues like calculating, visualizing, representing, merging, and analyzing model differences. The approach presented in this paper has demonstrated the idea to the representation of model differences. Future work will aim at realizing the approach for other transformation approaches. Furthermore, it is intended to elaborate appropriate approach and extend the DSL including other issues of the collaborative work.

4. REFERENCES

[1] Alan L. Glasser. The evolution of a Source Code Control System. *ACM SIGMETRICS Performance Evaluation Review*, Volume 7:Issue 3–4, November 1978.

[2] *James Rumbauch, Ivar Jacobson, and Grady Booch. The Unified Modeling Language Reference Manual.* USA, July, 2004.

[3] Y. Lin; J. Gray; and F. Jouault. DSMDiff: a differentiation tool for domain-specific models. *European Journal of Information Systems*, pages 16(4):349—-361, August 2007.

[4] Ebert, Jürgen; Riediger, Volker; Winter, Andreas. Graph Technology in Reverse Engineering, The TGraph Approach, GI, In: Gimnich, Rainer; Kaiser, Uwe; Quante, Jochen; Winter, Andreas (eds):. *10th Workshop Software Reengineering (WSR 2008)*, vol. 126(3):pp. 67–81, 2008.

[5] M. Alanen and I. Porres. Difference and Union of Models. *In P.Stevens, J.Whittle, and G. Booch, editors, Proc. 6th Int. Conf. on the UML*, volume 2863 of LNCS:pages. 2–17, October 2003.

[6] Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato,. Version Control with Subversion. *O'Reilly Media*, June 2004.

[7] Cicchetti, Antonio; Ruscio, Davide Di; Pierantonio, Alfonso. A Metamodel Independent Approach to Difference Representation. *Journal of Object Technology 6:9*, 2:165–185, October 2007.

[8] D. Varro, A. Balogh. The model transformation language of the VIATRA2 framework, July 2007.

[9] M. Herrmannsdoerfer. *Operation-based Versioning of Metamodels with COPE.* Vancouver, Canada, May 17 2009.

[10] F. Jouault and I. Kurtev. Transforming Models with ATL. *Springer-Verlag, In MoDELS Satellite Events*, volume 3844 of LNCS:pages 128—-138, September 5-9, 2005.

[11] T. Mens and S. Demeyer. Concurrent Versions Systems. *Springer*, 2008.

[12] P. Pietsch. *The SiDiff Framework. Technical report.* University of Siegen, Germany, March 13, 2009.

[13] G. Taentzer. AGG: A Tool Environment for Algebraic Graph Transformation. *Springer*, 1779:333–341, 2000.

[14] W. F. Tichy. RSC — a system for version control. *Software—Practice Experience*, Volume 15:Issue 7, July 1985.

[15] Timothy J. Grose, Gary C. Doney, Stephen A. Brodsky,. Mastering XMI. *OMG Press*, 2002.

[16] U. A. Nickel AND J. Niere AND A. Zündorf. Tool demonstration: The FUJABA environment. *Proc. of the 22nd International Conference on Software Engineering (ICSE), Limerick, Ireland*, 2000.