2012 28th IEEE International Conference on Software Maintenance (ICSM)

Understanding Model Evolution through Semantically Lifting Model Differences with SiLift

Timo Kehrer, Udo Kelter, Manuel Ohrndorf, Tim Sollbach
Software Engineering Group
University of Siegen, Germany
{*kehrer, kelter, mohrndorf, tsollba*}*@informatik.uni-siegen.de*

***Abstract*—In model-based software development, models are primary artifacts which iteratively evolve and which have many versions during their lifetime. A clear representation of the changes between different versions of a model is the key to understanding and successfully managing the evolution of a model-based system. However, model comparison tools currently available display model differences on a low level of abstraction, namely in terms of basic graph operations on the abstract syntax graph of a model. These low-level model differences are often hard or even impossible to understand for normal tool users who are not familiar with meta-models. In this paper we present SiLift, a generic tool environment which is able to semantically lift low-level differences of EMF-based models into representations of user-level edit operations.**

***Keywords*-model comparison; model difference; semantic lifting; difference presentation**

I. Introduction

Model-based software development has become a widespread approach for developing software in many application domains. Using platform-independent models as primary development artifacts reduces maintenance cost caused by the evolution of platforms, but does not suffice to successfully manage all aspects of the evolution, and actually creates new problems: When requirements change, models must change, too. Models therefore have many versions during their lifetime. A clear representation of the changes between different versions of a model is the key to understanding the evolution of a model-based system. Thus, high-quality model comparison tools are strongly required.

State-of-the-art model differencing tools compare models on the basis of their internal representation, i.e. an implementation of the abstract syntax graph (ASG). However, ASGs and their underlying meta-models can contain arbitrary design decisions and platform-specific details. Thus, even basic edit operations, which are available in model editors and which appear to be simple from a user's point of view, can lead to many low-level changes in the internal representation. Figure 1 shows an example of this for UML class models. Model A (upper-right window) is the base version, revision B (lower-right window) has been obtained by two editing steps:

1) The navigability of association `worksFor` is restricted to the association end `employee`.
2) The attribute `name` was moved to superclass `Person` using the refactoring "Pull Up Attribute".

The low-level changes induced by these edit operations are shown by the difference tree viewer on the left-hand side of Figure 1, which shows a screen-shot of the Difference Presentation UI of SiLift. In fact, the first editing step leads to the following set of low-level changes on the ASG:

- Reference `ownedAttribute` from class `Person` to property `employer` has been removed.
- Reference `ownedEnd` from association `worksFor` to property `employer` has been added.
- Reference `class` from property `employer` to class `Person` has been removed.
- Reference `owningAssociation` from property `employer` to association `worksFor` has been added.

Obviously, these low-level changes are not understandable for normal tool users who are not familiar with the details of meta-models and their platform-specific implementation. Low-level changes must be "semantically lifted", i.e. groups of low-level changes which represent the effect of a user-level edit operation must be identified and displayed appropriately.

This paper presents *SiLift*, an Eclipse-based framework for semantically lifting differences of EMF models. Comparison tool *users* are provided with a rich UI that allows them to interactively examine semantically lifted differences. Comparison tool *engineers* who have to adapt the lifting engine to a dedicated modeling language are supported with the generation and management of the necessary recognition rules. A download option and a demo video are provided at the project website available at http://pi.informatik.uni-siegen.de/Projekte/SiLift/.

II. Tool Overview

Figure 2 shows the four main steps of the comparison processing pipeline of SiLift:

(1) Initially, a **Matcher** searches for pairs of corresponding model objects which are considered to be "the same" in both models. The SiLift framework can be adapted to arbitrary matching engines, e.g. EMF Compare [5], SiDiff

978-1-4673-2312-3/12/$31.00

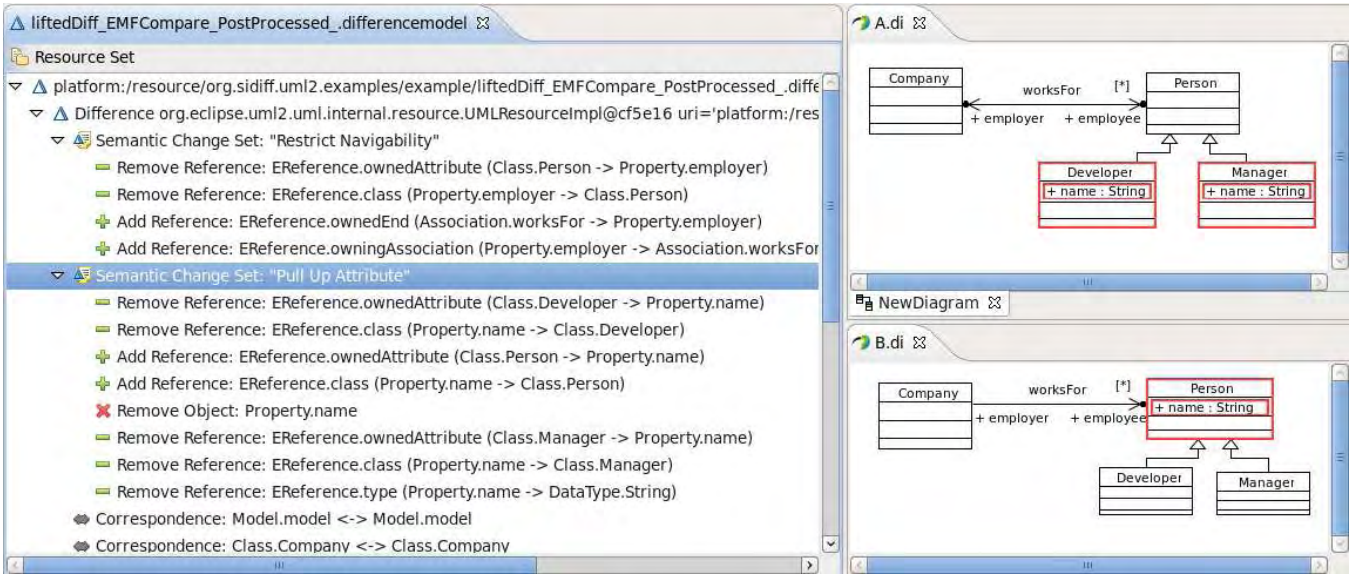

Figure 1. SiLift difference presentation UI: Low-level changes aggregated to semantic change sets

[9] or proprietary matching engines. The SiLift configuration provided at the accompanying web site contains (a) an adapter to the generic matching engine of EMF Compare and (b) a simple, but efficient signature-based matcher which establishes correspondences based on equal values of name attributes; these names are available for the most model element types in typical modeling languages.

A matching adapter for the SiDiff matching engine is available upon request. The SiDiff matching engine is not included in the product configuration provided at the web site due to license restrictions.

(2) Based on a given set of correspondences, a low-level difference is built by the generic **Difference Derivator** as follows: Objects and references not involved in a correspondence are considered to be deleted or created. A difference is represented as instance of the EMF-based difference model described in [10].

(3) The **Semantic Lifting Engine** identifies groups of low-level changes, referred to as *semantic change sets*, each one representing the effect of a user-level edit operation. The Semantic Lifting Engine is implemented using the rule-based model transformation framework EMF Henshin, the lifting algorithm is described in [10]. The transformation rules used by the semantic lifter are called *recognition rules*. They can be automatically derived from their corresponding *edit transformation rules*, each edit rule implementing a user-level edit operation. Tool-support on the "meta-level", i.e. the generation and management of rules, is considered in greater detail in Section III.

(4) Finally, the **Difference Presentation UI** serves as graphical interface and enables end-users to examine semantically lifted model differences. Difference presentation is considered in more detail in Section IV.

All SiLift components can be accessed via API. A `LiftingFacade` provides a set of convenience functions encapsulating the processing pipeline. For example, semantically lifted differences between two EMF models can be obtained by a single function call. This enables tool engineers to implement more complex version management functions such as patching or merging directly using high-level model differences.

III. Rule Management

A tool engineer who wants to adapt SiLift to a given modeling language has to provide a set of edit rules implementing the available user-level edit operations. We distinguish *atomic* edit operations and *complex* edit operations.

Atomic operations describe basic changes to a model. An atomic operation cannot be split into smaller modeling steps which maintain the model's consistency. Basically, there are four types of atomic edit operations: Element creations, element deletions, movements of elements and changes to elements' attribute values. Creations, deletions and movements have to take an element's mandatory context into account in order to preserve model consistency. For example, an UML association can only be created together with its association ends. Edit rules implementing atomic edit operations can be directly deduced from a given metamodel using an Edit Rule Generator (s. Figure 2).

A composite operation consists of a set of other atomic and/or complex operations carrying out smaller modifications. For example, the refactoring "Pull Up Attribute" for UML class models (cf. Section IV) consists of an attribute movement from a subclass to its superclass and an arbitrary number of equivalent attribute deletions from all of the sub-

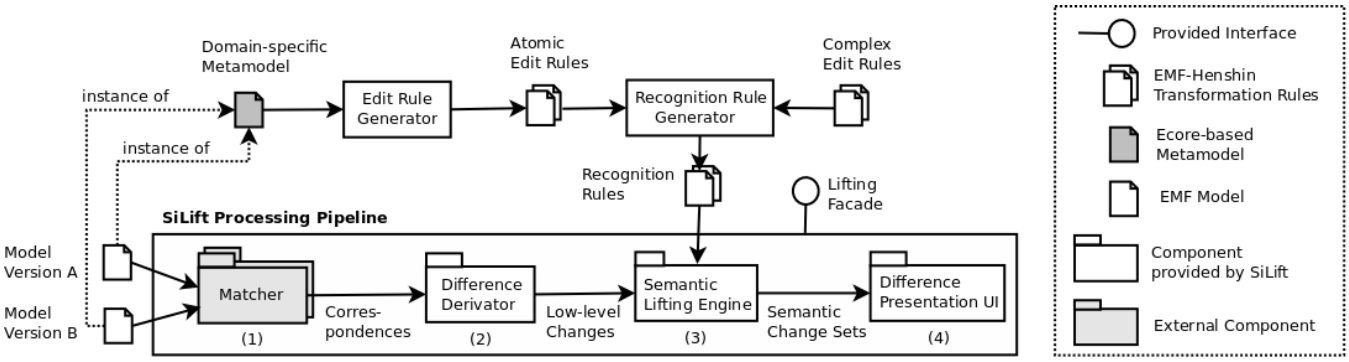

Figure 2. Tool Overview

class' neighbours in terms of the inheritance tree. Complex edit rules have to be designed manually.

Generated and manually implemented edit rules can be transformed into their corresponding recognition rules. The conceptual details of this transformation are described in [10]. Edit and recognition rules can be synchronized and versioned using the graphical Rule Manager (s. Figure 3). If desired, a priority value can be assigned to each recognition rule. Thus, ambiguities can be resolved if several semantic change sets contain exactly the same low-level changes. This type of ambiguity can not be resolved by the postprocessing algorithm described in [10]. For example, recognizing navigability restriction of UML association ends should be usually assigned a higher priority value than recognizing the respective Property movement from Association to Class.

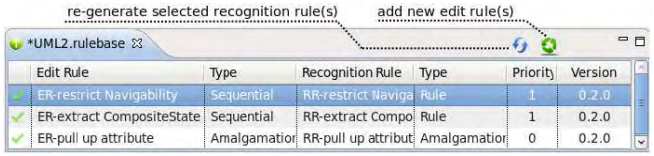

Figure 3. Management of edit and recognition rules

IV. Difference Presentation

Comparison tool users are provided with a rich UI that presents semantically lifted differences. Figure 1 illustrates the Difference Presentation UI of SiLift by means of the UML class model example which was introduced in Section I. On the left-hand side of the Presentation UI, a difference is shown in a tree-based viewer. The tree viewer serves as the main control window in order to interactively explore the difference. In the example illustrated in Figure 1, the tree viewer contains two semantic change sets "setNavigability" and "Pull Up Attribute". Each of them represents the related edit operations which have been applied to model version A. The low-level changes which are grouped by the semantic change sets are presented hierarchically. Additionally, the correspondences obtained from the matcher are displayed by the tree viewer.

On the right-hand side, the UI contains the native graphical editor windows which were used to create model versions A and B. Thus, models are displayed in the notation a user is familiar with. If the notation is graphic, the original layout of the diagrams is preserved. If an operation displayed by the tree viewer is selected then all involved diagram elements in the editor windows are highlighted. In this way, the context of a model change or the elements involved in a correspondence are visualized. In Figure 1, the selection of the semantic change set "Pull Up Attribute" causes a highlighting of the relocated name attributes and their respective class contexts.

The tree viewer on the left-hand side and the native editors on the right-hand side are loosely coupled via the Eclipse Selection Service which notifies any registered listener about selection changes induced by a selection provider (s. Figure 4). We have implemented a selection listener that decorates shapes and connections serving as diagram elements in the Eclipse Graphical Modeling Framework (GMF).

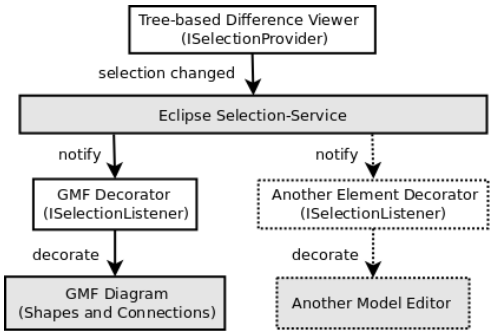

Figure 4. Difference presentation architecture

This solution is generic in the sense that it works for all GMF-based editors. In fact, the models which are shown in Figure 1 were created with the Eclipse-based UML2 modeling tool Papyrus, which uses GMF for graphical diagram modeling. However, the element highlighting facility can be implemented for editors based on any technology as long as an external interface is available. Otherwise, the difference

tree viewer can also be used in "standalone mode". Of course, element highlighting in the editor windows on the right-hand side is not limited to graphical editors.

V. Related Work

Difference tools for models have been addressed by a large number of publications recently [3]. One class of approaches which dates back to 1992 [13] is based on logging editing commands in syntax-based editors, e.g. Fujaba [14] and elsewhere [8], [11]. Logging-based approaches have the big advantage that change logs can be obtained directly on the level of user commands or at least on the level of internal edit operations, of course also on the level of basic storage operations. However, documents must not be modified by tools which do not carefully maintain the change logs, change logs must never be deleted etc.; thus logging-based approaches imply closed environments. Moreover, these approaches cannot compare documents which are not revisions of each other, e.g. re-engineered models or the output of model transformers. Thus, logging-based approaches are not a general solution of the problem.

Most approaches compare models based on their actual states. Only a few generic approaches to model comparison have been proposed so far, an overview can be found in [9]. However, model comparison tools currently available, e.g. [16], [1], usually just cover the first two steps of the processing pipeline shown in Figure 2. Support for the semantic lifting of model differences is limited to the following approaches.

Könemann [12] considers differences as patches which shall be applied to a model which differs significantly from the original base model from which a difference has been computed. The "literal" difference is thus transformed into a more "fuzzy" specification of the desired changes, i.e. similar atomic changes are grouped using name patterns or OCL queries. Essential steps of this fuzzification rely on user interaction: the correctness of introduced abstractions must be checked and initial drafts proposed by the system must be corrected manually. In comparison to [12], change set recognition rules for the SiLift semantic lifting engine are systematically derived from specifications of edit operations, and the application of these rules is fully automated.

A development tool named By-Example Operation Recorder [2] enables users to specify composite edit operations "by example". Basically, a tool engineer creates a typical model as base revision and edits the model to produce the effect of the composite operation. The Operation Recorder finally delivers a difference pattern which represents the effect of the composite operation. They suggest such difference patterns to be used for annotating model differences, thus lifting the difference to the level of user operations. However, it remains unclear how instances of composite operations are found in a given difference, which matching engine is assumed and how ambiguities are handled if the same atomic changes are possibly involved in different composite operations.

A UML2-specific extension to the Generic Diff Engine of EMF Compare is proposed in [15]. Some low-level changes induced by UML-specific edit operations are aggregated into high-level ones. However, the recognition code to identify "UML-level change operations" from the "generic EMF-level change operations" must be implemented manually for each UML edit operation that shall be supported. No methodology for tool engineers is provided, thus no general support for adopting the approach to other model types is available.

Acknowledgment

This work was supported by Deutsche Forschungsgemeinschaft under grant KE 499/7-1.

References

[1] Brand, Mark van den; Protic, Zvezdan; Verhoeff, Tom: RCVDiff - a stand-alone tool for representation, calculation and visualization of model differences; p.96-101 in: Proc. Workshop on Models and Evolution. Oslo, Norway; 2010

[2] Brosch, P.; Langer, P.; Seidl, M.; et al.: An Example is Worth a Thousand Words: Composite Operation Modeling By-Example; p.271-285 in: Proc. Intl. Conf. Model Driven Engineering Languages and Systems 2009, Denver; LNiCS 5795, Springer; 2009

[3] Bibliography on Comparison and Versioning of Software Models; http://pi.informatik.uni-siegen.de/CVSM

[4] Eclipse Modeling Framework; http://www.eclipse.org/emf

[5] EMF Compare; http://www.eclipse.org/emf/compare

[6] Henshin; http://www.eclipse.org/modeling/emft/henshin/

[7] Graphical Modeling Framework; http://www.eclipse.org/modeling/gmp/

[8] Herrmannsdörfer, M.; Kögel, M.: Towards a Generic Operation Recorder for Model Evolution; p.76-81 in: Proc. Workshop on Model Comparison in Practice, Malaga; 2010

[9] Kehrer, T.; Kelter, U.; Pietsch, P., Schmidt, M.: Adaptability of Model Comparison Tools; to appear in: Proc. Conf. Automated Software Engineering (ASE 2012); 2012

[10] Kehrer, T.; Kelter, U.; Taentzer, G.: A Rule-Based Approach to the Semantic Lifting of Model Differences in the Context of Model Versioning; p.163-172 in: Proc. Conf. Automated Software Engineering (ASE 2011); 2011

[11] Kögel, M.: TIME - Tracking Intra- and Inter-Model Evolution; p.157-164 in: Software Enineering 2008 - Workshopband, 18.-22.. Februar 2008 in München, Germany; LNI, P-122, Bonner Köllen Verlag; 2008

[12] Könemann, P.: Capturing the Intention of Model Changes; p.108-122 in: Proc. Conf. Model Driven Engineering Languages and Systems 2010, Oslo; 2010

[13] Lippe, E.; Oosterom, N.: Operation-based Merging; p.78-87 in: ACM SIGSOFT Software Eng. Notes 17:5; 1992

[14] Schneider, C.; Zündorf, A.; Niere, J.: CoObRA - a small step for development tools to collaborative environments; Workshop on Directions in Software Engineering Environments at ICSE 2004, Edinburgh; 2004

[15] Extensions to EMF Compare; http://bit.ly/iNeV3q

[16] Xing, Zhenchang: Model Comparison with GenericDiff; in: Proc. 25th IEEE/ACM International Conference on Automated Software Engineering (ASE'10); ACM; 2010