Automatic Generation of Consistency-Preserving Edit Operations for MDE Tools

Michaela Rindt, Timo Kehrer, Udo Kelter

Software Engineering Group
University of Siegen
{mrindt,kehrer,kelter}@informatik.uni-siegen.de

Abstract. Many tools for Model-Driven Engineering (MDE) which are based on the widespread Eclipse Modelling Framework (EMF) [4] are developed for single tasks like e.g., generating, editing, refactoring, merging, patching or viewing of models. Thus, models are oftentimes exchanged in a series of tools. In such a tool chain, a grafical model editor or viewer usually sets the degree of well-formedness of a model in order to visualize it. Well-formedness rules are typically defined in the meta-models, yet not all tools take them into account. As a result, a model can become unprocessable for other tools. This leads to the requirement, that all tools should be based on a common definition of minimum consistency.
An obvious solution for this challenge is to use a common library of consistency-preserving edit operations (CPEOs) for models. However, typical meta-models lead to a large number of CPEOs. Manually specifying and implementing such a high number of CPEOs is hardly feasible and prone to error. This paper presents a new meta-tool which generates a complete set of CPEOs for a given meta-model. We have successfully integrated the generated CPEOs in several developer tools. The video http://youtu.be/w31AcMOd83Y demonstrates our meta-tool in the context of one of our developer tools.

1 Introduction

Model-Driven Engineering (MDE) must be supported by tools which can edit or refactor (e.g., [2]), generate (e.g., [8]), patch or merge models (e.g., [6]). These tools are typically based upon the Eclipse Modeling Framework (EMF) [4], in which a model is represented as an Abstract Syntax Graph (ASG). Frameworks such as EMF provide basic API methods to edit the ASG of a model, e.g. creating, deleting or updating single objects or attributes. However, editing ASGs with such low-level operations can violate consistency constraints on the ASG defined in a meta-model. The resulting inconsistent ASGs cannot be processed and graphically visualized by most MDE tools. In order to solve this problem, all model editing tools should use a common library of *consistency-preserving edit operations (CPEOs)*. These CPEOs must be tailored to the relevant meta-model and its constraints. Unfortunately, complete sets of CPEOs can be quite large for comprehensive meta-models such as the UML [11] meta-model. Obviously, the

manual implementation of a large number of CPEOs, e.g., as code or executable transformations, is not only tedious, but also very error-prone.

The main contribution of this paper is a meta-tool called SERGe (SiDiff Edit Rule Generator) which generates a complete set of executable CPEOs for a given meta-model. The generated sets of CPEOs can be integrated by tool developers into an MDE environment as illustrated in Figure 1. In this example, a model generator integrates the functionality of SERGe to initially generate a set of CPEOs. Afterwards, the model generator algorithm can execute these CPEOs to generate models. Moreover, the generated CPEOs comprise a common library which can be reused by further tools, e.g., a model refactor tool.

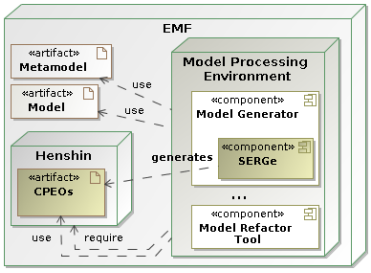


Fig. 1. Deployment Diagram showing SERGe and CPEO integration

The generation process for a set of CPEOs is fully automated and meta-model independent. SERGe is based on EMF. The generated CPEOs use EMF Henshin [5] as the transformation language and require the Henshin interpreter as the execution platform. Henshin transformation rules are in-place transformations and can contain model patterns to be found and preserved, deleted or created and also to be forbidden or required. Some consistency criteria are already enforced by EMF, e.g., type conformance, guaranteeing at most one container for each model element or a consistent handling of opposite references. With CPEOs generated by SERGe we can extend this list by (a) the preservation of multiplicity constraints and (b) the prevention of containment hierarchy cycles. The generated CPEO sets are complete in the sense that any change between two consistent models can be expressed using these CPEOs. These types of consistency constraints are sufficient to be able to graphically display models. We are not aware of an existing model editor which is usable in combination with other EMF based MDE tools and which enforces stronger consistency constraints. There can be more advanced constraints (i.e., OCL Constraints) inside a meta-model. However, they are typically not enforced by model editors and thus are not covered with SERGe so far.

SERGe provides a variety of optional configuration settings to tailor the generation process, e.g., whether to generate CPEOs for supertypes instead of for each subtype. The former will decrease the number of generated CPEOs heavily. One can also enable or disable the kinds of CPEOs (create, move, etc.) that should be generated. These are just a few configurations that are possible. SERGe has already been used extensively in different research projects, e.g., the SMG (SiDiff Model Generator) [8], SiLift [6, 10] for difference recognition between models and patching of models, and others [7]. Further possible use-cases can be tools for merging, refactoring or checking of models. More information and an example set of CPEOs can be found at the SERGe project website [9].

2 Consistency-preserving Edit Operations (CPEOs)

The easiest way to modify the Abstract Syntax Graph (ASG) of a model is to use *basic graph operations* e.g., creating or deleting single model objects. However, basic ASG operations do not consider well-formedness rules (e.g., multiplicity constraints) as defined by the meta-model. Hence, they can lead to inconsistent ASGs, which cannot be processed by other tools.

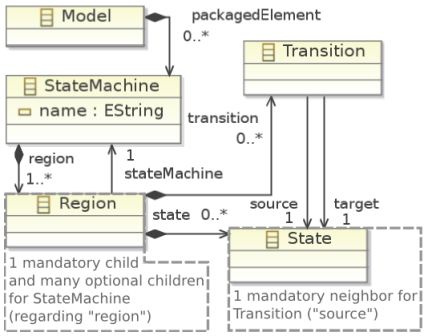
As an example, we use simplified state machines with a meta-model as shown in Figure 2(a). A `StateMachine` object must have at least one child object of type `Region`, s. the multiplicity constraint of $[1..*]$ of the *containment* reference `region`. A basic ASG operation which creates only a single `StateMachine` object violates this constraint. A CPEO on the other hand will create a `StateMachine` object together with a contained, mandatory child object of type `Region`.

A CPEO usually comprises several basic ASG operations, but at least those which are required to implement a consistency-preserving editing behavior. Figure 2(b) shows the CPEO mentioned above as an EMF Henshin [5] transformation rule named 'createStatemachineInModel'. Another example is provided in Figure 2(d). The example CPEO rule has a few input and output parameters: e.g., `Selected` is a placeholder for an input model object which defines the context for the transformation application. Figure 2(d) depicts the changing of an old targeted State object to a new target State in the context of a Transition. This operation contains two ASG operations, notably the deletion of an old reference `target` and the creation of a new reference `target`.

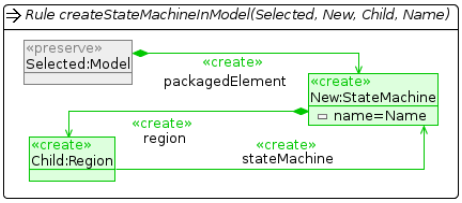
3 Generation of CPEOs

Prior to the generation phase, the meta-model is analyzed to identify the relationships between classifiers. This is done by considering incoming references of each classifier and the complete inheritance hierarchy. The source of a reference can either be a parent context or a neighbor context. This depends on the nature of a reference which is either *containment* in the first or *non-containment* in the latter case. Analogously, the target of a reference can either be identified as a 'child' or a 'neighbor'. Naturally, opposite references (e.g., `region` and `stateMachine` in the example) have to be considered together. Otherwise invalid CPEOs could be generated. An invalid operation would be the change of the reference target `stateMachine` without also changing its opposite, namely the containment reference region

Multiplicities of a reference (i.e., the upper bound (ub) and lower bound (lb)) are classified by one or more of the invariant groups shown in Figure 2(c). The meta-model analysis categorizes each relationship by considering each multiplicity invariant, which can be found on a reference. It determines if a target of a reference needs **mandatory** objects. This is the case if the reference multiplicity is classified as *required*. In a relationship between model elements, there can also exist **optional** objects. This is the case if the reference is attached with a *many* multiplicity classification. Naturally, these classifications can both apply



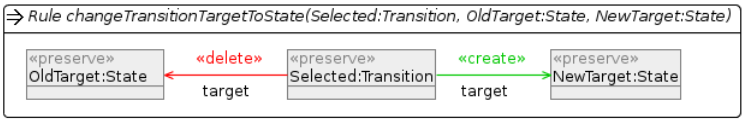
(a) simple state machine meta-model



(b) CPEO 'createStateMachineInModel'

classification	invariant group ($lb \leq ub$)
required	$lb > 0$
bounded	$ub < \infty$
fixed	$lb = ub$
many	$(ub - lb >= 1) \vee (ub = \infty)$

(c) multiplicity invariant classification



(d) CPEO 'changeTransitionTargetToState'

Fig. 2. Generation details

to one reference, e.g., for [1..∗] (see Figure 2(a)). This identification allows the generation algorithm to decide if a CPEO for the creation of an object may be generated or if this creation may only happen in the context of another CPEO.

During the generation phase every classifier in the given meta-model is visited. By means of the previously analyzed relationships and attributes of each considered classifier, SERGe determines which CPEO kinds will be generated for which classifier, reference or attribute. The starting point for each decision is the nature of a reference (i.e., being *containment* or *noncontainment*). [1] The following CPEO kinds can be generated depending on the occurring multiplicity invariants: creation/deletion of elements, `adding`/`removing` neighbors, `setting`/`unsetting`/`changing` of single neighbors or (default) attribute values or `moving` of children between different contexts. Mandatory children and neighbors of elements are integrated recursively inside a CPEO.

The generated CPEOs can also contain precondition checks to avoid falling below *required* multiplicities or exceeding *bounded* multiplicities. This is realized with Henshin *Positive Application Conditions (PAC)* and *Negative Application Conditions (NAC)*.

[1] We assume attributes can be handled equally to non-containment references.

4 Related Work

There are several approaches to generate executable edit operations for models beyond basic ASG edit operations. The closest approaches to ours are [1, 3].

Ehrig and Taenzer [3] address the problem to generate correct instances of a given meta-model. In such a context, only edit operations which create model elements are needed. Edit operations which delete or modify models are not provided. The problem that mandatory components cannot be simply deleted, but only be replaced, is not addressed here. The generated sets of edit operations are thus not complete in our sense. Moreover, the final result of the instance generation process must conform to the meta-model; here intermediate and inconsistent states can occur and need to be repaired afterwards. Our CPEOs on the other hand never produce inconsistent intermediate states when applied; i.e., CPEOs preserve the consistency by-construction.

Alanen and Porres [1] proposes to first convert a model into a string representation, then edit the model using a syntax-directed editor, and finally to convert it back to an ASG-based representation. Although basic consistency constraints can be preserved this way, this process is not very convenient, especially in the case of visual models.

To our best knowledge, the coverage of consistency constraints, configurability and completeness of the generated sets is not met by any other existing meta-tool to generate edit operations.

References

1. Alanen, M.; Porres, I.: A relation between context-free grammars and meta object facility metamodels; Technical Report 606, TUCS Turku Center for Computer Science; 2003
2. Arendt, T.; Taentzer, G.: A tool environment for quality assurance based on the Eclipse Modeling Framework; p.141-184 in: Automated Software Engineering 20(2); 2013
3. Ehrig, K.; Küster, J.M.; Taentzer, G.; Generating instance models from meta models; p.479-500 in: Software and Systems Modeling 8(4); 2009
4. EMF: Eclipse Modeling Framework; 2014; http://www.eclipse.org/emf;
5. EMF Henshin; 2014; http://www.eclipse.org/modeling/emft/henshin/
6. Kehrer, T.; Kelter, U.; Taentzer, G.: Consistency-Preserving Edit Scripts in Model Versioning; p.191-201 in: Proc. 28th IEEE/ACM Intl. Conf. Automated Software Engineering (ASE 2013); 2013
7. Kehrer, T.; Rindt, M.; Pietsch, P; Kelter, U.: Generating Edit Operations for Profiled UML Models; p.30-39 in: Proc. Models and Evolution (ME 2013); 2013
8. Pietsch, P.; Shariat Yazdi, H.; Kelter, U.: Generating Realistic Test Models for Model Processing Tools; p.620-623 in: Proc. 26th IEEE & ACM Inter. Conf. Automated Software Engineering (ASE 2011); ACM; 2011
9. SERGe; Project Page; 2014; http://pi.informatik.uni-siegen.de/Projekte/-SERGe.php
10. SiLift project website; http://pi.informatik.uni-siegen.de/Projekte/SiLift
11. Unified Modeling Language: Superstructure, Version 2.4.1; OMG, Doc. formal/2011-08-05; 2011