Integrating the Specification and Recognition of Changes in Models

Timo Kehrer, Udo Kelter
Praktische Informatik
Universität Siegen
{kehrer,kelter}@informatik.uni-siegen.de

Gabriele Taentzer
Fachbereich Mathematik und Informatik
Philipps-Universität Marburg
taentzer@mathematik.uni-marburg.de

Abstract

Model-based software development has become a widely accepted approach in application domains where software is long-living and must be maintainable. Models are subject to continuous change and have many versions during their lifetime. The specification and recognition of changes in models is the key to understand and manage the evolution of a model-based system. However, model transformation and model versioning tools currently available are based on low-level operations on models, which are sometimes incomprehensible for users. Thus, both classes of tools should be better integrated and extended to supporting high-level edit operations, e.g. refactoring operations.

1 Introduction

Development technologies for long-living software systems are faced with the huge challenge of supporting the evolution of requirements and platforms. In model-based software development (MBSE), the use of platform-independent models as primary development artifacts reduces maintenance cost caused by the evolution of platforms. However, MBSE does not suffice to successfully manage all aspects of evolution, and actually creates new problems: When requirements change, models must change, too. Models of long-living software systems therefore have many versions during system lifetime. Traditional line-oriented differencing and merging approaches, which are successfully used for the version management of program source code, can usually not be applied to software models. This has triggered a lot of research into model versioning recently [2].

Being able to specify changes between revisions in a precise and meaningful way is of primary importance to understand and plan the evolution of a model-based system. Specifications of changes in models between versions play (a) a *prescriptive* role, i.e. as specification of modifications to be performed on an existing model version, and (b) a *descriptive* role as a means to describe the observed difference between two versions, notably past changes in the history of a model.

Meaningful specifications of changes must reflect the way in which models are edited by users, which particularly includes complex editing operations, a concrete example is provided in Section 2. These complex modifications are already supported by model transformation and refactoring tools [1]. However, their recognition is an open problem which is briefly introduced in Section 3. Thus, typical model versioning tasks such as patching and merging still operate on the basis of low-level changes; they should be lifted to the level of user edit operations. In other words, model editing and model versioning tools must be integrated in the sense that they are based on the same set of edit operations which are applicable from a user's point of view. A summary of our research agenda is given in Section 4.

2 Editing of Models

Changes in models can only be specified precisely if a runtime representation of models is available. To that end, the concept of metamodeling was established in the modeling community. Models are considered as abstract syntax graphs (ASG), metamodels define the types of nodes and edges of the ASG. Metamodels do not directly specify behavior, i.e. *edit operations* which can modify models. Many basic model editing operations, e.g. the creation of a model element of type T or the setting of a property of an element, can be directly deduced from a given metamodel.

However, basic model edit operations can violate well-formedness rules. The UML metamodel, for instance, defines many mutually dependent data items in an ASG and related consistency constraints. Project-specific modeling guidelines can define further constraints. Then a consistency-preserving change of data items requires several basic edit operations, i.e. a complex edit operation. Refactoring operations and tool commands lead to further complex edit operations. Figure 1 shows a sample Simulink block diagram that has been modified as follows: The linear function $f(x) = mx + c$, which processes the observed sine signal, is extracted into a separate subsystem. This effect can be achieved by *one* user-level command, namely the edit operation "Create Subsystem".

3 Differencing of Models

Some approaches to model differencing are based on the logging of edit commands in editors. However, these approaches requires closed environments and do

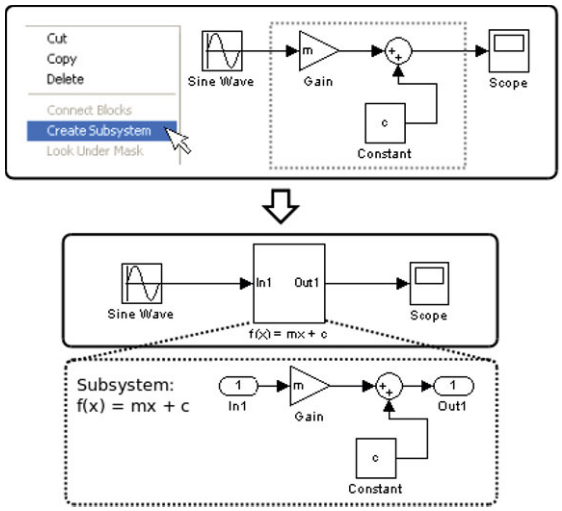

Figure 1: Creation of a subsystem in Matlab/Simulink

not provide a general solution to the problem [6].

State-based approaches compare models on the basis of their state and usually work on an ASG representation of the models [5]. Initially, a matching procedure identifies corresponding model elements and relationships in both models, i.e. corresponding nodes and edges in their ASGs. Model elements and relationships not involved in a correspondence are considered to be deleted or created; these insertions and deletions form a *low-level difference*. Low-level differences often contain pseudo changes which do not make sense from a users' point of view [4]. Developers perceive models in their external, typically graphical representation, and prefer changes to be explained in terms of conceptually meaningful edit operations.

A first approach to overcome this problem is presented in [3]: A *semantic lifter* identifies groups of low-level changes which implement an edit operation. However, this approach does not cover more complex edit operations yet. For example, the subsystem extraction of Figure 1 still leads to several low-level changes which can be briefly summarized as follows: Firstly, the created subsystem is embedded into the parent system by a subsystem block serving as black-box and providing the connector ports "In1" and "Out1". Secondly, within the implementation of the subsystem, the blocks "In1" and "Out1" provide the necessary connection points for incoming and outgoing data flows. Finally, the blocks (gain, constant and add-operator) and respective data flows realizing the linear function are finally relocated into the new subsystem.

4 Integrating Specification and Recognition of Model Changes

Technologies for editing and differencing models must be based on the same set of editing operations if one wants to be able to repeat observed changes (as a patch) and if model comparisons shall report the same complex changes which were actually performed. This is trivial in case of basic graph operations. It is a big challenge if complex edit operations are to be supported.

[3] presents a first approach for lifting model differences to representations of simple edit operations. This approach can be extended towards more complex edit operations. Formalisms for defining complex operations and methods to recognize the resulting changes mutually depend on each other. Our research agenda thus comprises the following sub-goals:

1. Identifying *meaningful complex edit operations* in models. These operations depend a lot on the application domain and the types of models which are used, they must be defined individually for each application domain. Model types used for embedded systems in domains like automation engineering etc., e.g. block diagrams, present a particular challenge due to their size and recursive structure.

2. Providing a comprehensive *specification language for complex edit operations* and developing the underlying theoretical background.

3. Developing *model comparison* algorithms which recognize complex edit operations performed between two revisions of a model.

4. Extending algorithms for *model merging and patching*, including conflict and dependency detection, and model history analysis to support complex edit operations.

References

[1] Arendt, T.; Biermann, E.; Jurack, S.; Krause, C.; Taentzer, G.: Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations; in: Proc. MoDELS 2010; Springer, LNCS 6394; 2010

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

[3] 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. ASE 2011; ACM; 2011

[4] Kelter, U.: Pseudo-Modelldifferenzen und die Phasenabhängigkeit von Metamodellen; p.117-128 in: Proc. SE 2010; LNI 159; 2010

[5] Kolovos, D.S.; Ruscio, D.D.; Pierantonio, A.; Paige, R.F.: Different Models for Model Matching; p.1-6 in: Proc. CVSM 2009; IEEE; 2009

[6] Küster, J.M.; Gerth, C.; Förster, A.; Engels, G.: Detecting and Resolving Process Model Differences in the Absence of a Change Log; p.244-260 in: Proc. BPM 2008; Springer, LNCS 5240; 2008