TReMer: A Tool for Relationship-Driven Model Merging

Mehrdad Sabetzadeh and Shiva Nejati

Department of Computer Science, University of Toronto,
Toronto, ON M5S 3G4, Canada.
Email: {mehrdad,shiva}@cs.toronto.edu

1 Introduction

Model management is a crucial activity in large-scale software development where development tasks are often distributed over different teams. To support flexible, coordinated work, these teams need to maintain partial models of the overall system, and understand the relationships between these models. Models of a proposed system may be manipulated in various ways, and the results of these manipulations may be used by analysts to evolve the models. Model management aims to keep track of the relationships between a set of models as they evolve, and to describe the manipulations performed over them in terms of a set of predefined operators. Bernstein [2] identifies a number of useful operators on models, including *Match*, for finding correspondences between models, *Diff*, for finding differences between models, and *Merge*, for combining a set of models. Easterbrook et al. [3] extend these with several complementary operators including *Slice*, for producing a projection of a model based on a given criterion, and *Split*, for retrieving the models involved in building a composite one.

In conventional approaches to software development, models and programs are treated as textual artifacts and are managed via version control systems, supported by a number of relatively simple textual operations, including cut and paste, and text based differencing. In model-driven software development [12], artifacts are expressed in well-defined notations, such as those comprising UML. The richer semantic basis of these notations gives rise to more sophisticated model operations, and hence a greater management challenge.

In this extended abstract, we describe a tool, TReMer, for performing the merge operation in a model-driven development setting. Merge is arguably one of the most important model management operations and is useful as a way of consolidating a set of models to gain a unified perspective, to understand their interactions, or to perform various types of end-to-end analysis over them.

TReMer draws on the theory developed in our earlier work [8, 9, 13, 6, 3] where we describe how a set of models can be merged w.r.t. known or hypothesized relationships between them. We treat model relationships as explicit artifacts. This treatment offers two major advantages: Firstly, it makes our tool adaptable to different modelling domains. In particular, as we will argue in the next section, explicit relationships make it possible to provide a unified merge framework for both structural and behavioural modelling domains. Secondly, explicit relationships facilitate the exploration of merge alternatives by allowing users to articulate each alternative in a precise way.

A notable feature of TReMer is its ability to handle models that may be incomplete and inconsistent. Due to lack of space, we do not discuss this feature here, and instead, refer the reader to our previous work [11, 10] where a detailed exposition of the subject has been provided.

2 Tool Overview

Fig. 1 shows an overview of the methodology employed in TReMer for model merging. Given a set of models, the tool allows users to graphically explore and identify the relationships between them. The input models and their relationships are supplied to the merge component which in turn generates a merged model by applying a domain-specific merge algorithm. The merged model is then presented to the user for further analyses. These analyses may lead to the discovery of new relationships or the invalidation of some of the existing ones. The user may then start a new iteration by revising their merge hypothesis (or creating a new hypothesis) and following the subsequent activities.

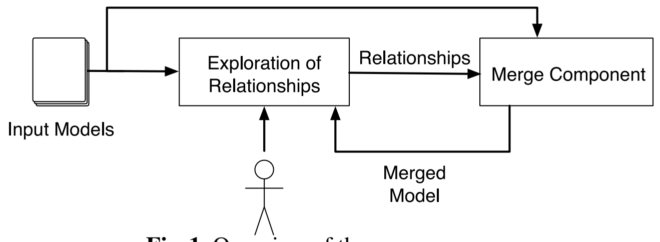

Fig. 1. Overview of the merge process.

As suggested by Fig. 1, our tool consists of two main parts: (1) a front-end that allows users to explore and specify relationships between models; and (2) a library of merge algorithms. Our current merge library supports two general classes of models: structural, e.g., Entity-Relationship (ER) models, and behavioural, e.g., state-machine models. The key factor that allows us to unify merging of these two seemingly different classes of models is the treatment of relationships as first class objects. Such treatment makes it possible to define an abstract unified interface for the merge process. Each domain-specific merge algorithm is then implemented as a *plugin* conforming to this unified interface.

To capture relationships between structural models, we use submodels, also referred to as *connectors* [9]. Connectors, which are usually specified by domain experts, describe the common parts between a set of models. Our structural merging algorithm is based on a category-theoretic concept called *colimit* [1]. Colimits provide a very high-level and yet powerful machinery for merging [5]. Intuitively, computing the colimit yields a new model combining all models w.r.t. their correspondences as described by a set of given relationships. Several examples on merging structural models are available in [9, 8, 11].

In behavioural modelling, a relationship is specified as a binary relation between the states of the input models. The behavioural merging algorithm for a pair of models is defined as their *common refinement* [13, 6]. Common refinement captures the "more complete than" relation between behavioural model pairs and has proven to be a suitable notion for defining behavioural merges. In order to compute a common refinement of two state machines, we first need a relationship between the states of the two models. Such a relationship effectively allows us to describe the correspondences between the two models and compute their merge accordingly. Relationships between state-machines can be computed automatically when input models are behaviourally consistent, or can be provided by domain experts, otherwise. For examples of behavioural merging, the reader can refer to [13, 6, 4].

For future versions of the tool, we plan to carry out a systematic study of the domains where model merging plays an important role, and use the results of this study to build a more comprehensive library of merge algorithms. Further, we plan to extend our work to other model management operators discussed in [3]. A major part of our ongoing research in this direction is focused on the Match operator which provides semi-automated assistance for finding model relationships. Some initial work in this direction has been reported in [7].

Acknowledgment. We thank Marsha Chechik and Steve Easterbrook for their help and support throughout this work.

References

1. M. Barr and C. Wells. *"Category Theory for Computing Science"*. Les Publications CRM Montréal, Montreal, Canada, 3rd edition, 1999.
2. P. Bernstein. "Applying Model Management to Classical Meta Data Problems". In *Proceedings of the 1st Biennial Conference on Innovative Data Systems Research*, pages 209–220, 2003.
3. G. Brunet, M. Chechik, S. Easterbrook, S. Nejati, N. Niu, and M. Sabetzadeh. A manifesto for model merging. In *Wkshp. on Global Integrated Model Management (GaMMa)*, 2006.
4. G. Brunet, M. Chechik, and S. Uchitel. Properties of behavioural model merging. In *Proceedings of Formal Methods (FM'06)*, 2006. To appear.
5. J. Goguen. "A Categorical Manifesto". *Mathematical Structures in Computer Science*, 1(1):49–67, 1991.
6. S. Nejati and M. Chechik. "Let's Agree to Disagree". In *Proceedings of 20th IEEE International Conference on Automated Software Engineering (ASE'05)*, pages 287 – 290, 2005.
7. S. Nejati, M. Sabetzadeh, M. Chechik, and S. Easterbrook. "Identifying and Representing Requirements Variability in Families of Reactive Software". University of Toronto, Technical Report CSRG-538, 2006.
8. M. Sabetzadeh and S. Easterbrook. "Analysis of Inconsistency in Graph-Based Viewpoints: A Category-Theoretic Approach". In *Proceedings of the 18th IEEE International Conference on Automated Software Engineering*, pages 12–21, October 2003.
9. M. Sabetzadeh and S. Easterbrook. "An Algebraic Framework for Merging Incomplete and Inconsistent Views". In *13th IEEE International Requirements Engineering Conference*, pages 306–318, September 2005.
10. M. Sabetzadeh and S. Easterbrook. iVuBlender: A tool for merging incomplete and inconsistent views. In *13th IEEE International Requirements Engineering Conference*, pages 453–454, September 2005.
11. M. Sabetzadeh and S. Easterbrook. View merging in the presence of incompleteness and inconsistency. *Requirements Engineering*, 11(3):174–193, 2006.
12. T. Stahl, M. Vylter, and K. Czarnecki. *Model-Driven Software Development*. John Wiley and Sons, 2006.
13. S. Uchitel and M. Chechik. "Merging Partial Behavioural Models". In *Proceedings of 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 43–52, November 2004.