A Relationship-Driven Framework for Model Merging *

Mehrdad Sabetzadeh Shiva Nejati Steve Easterbrook Marsha Chechik
Department of Computer Science, University of Toronto
Toronto, ON M5S 3G4, Canada.
Email: {mehrdad,shiva,sme,chechik}@cs.toronto.edu

Abstract

A key problem in model-based development is merging a set of distributed models into a single seamless model. To merge a set of models, we need to know how they are related. In this position paper, we discuss the methodological aspects of describing the relationships between models. We argue that relationships between models should be treated as first-class artifacts in the merge problem and propose a general framework for model merging based on this argument. We illustrate the usefulness of our framework by instantiating it to the state-machine modelling domain and developing a flexible tool for merging state-machines.

Keywords: Model-Based Development, Distributed Modelling, Model Merging.

1 Introduction

An effective way to manage complexity and promote flexibility in large-scale model-based development is to describe a proposed system using loosely-coupled partial models, known as *views* [10]. Each view considers the system from a particular angle, making it possible to scope the amount of information that analysts need to deal with at any given time [9]. Further, by providing means for separating the contributions of different sources, views allow stakeholders to express and maintain their individual perspectives on the system [8].

A key problem in view-based modelling is *model merging*. From time to time, we need to integrate information from several models into a single one in order to gain a unified perspective, to understand interactions between models, or to perform various types of end-to-end analysis [17]. Model merging spans several application areas. In information systems, model merging is an important step during conceptual database design for constructing an abstract schema that captures the data requirements of all the involved participants [4]. In ontology research, model merging may be employed to build a global ontology from a set of local ontologies originating from different communities [11]. Software engineering deals extensively with model merging – several papers study the subject in specific domains including early requirements [17], use-cases [15], class diagrams [2, 21], software architectures [1], state-machines [16, 19, 14], and sequence diagrams [20, 7].

Model merging frameworks differ in several factors including the notations they support, the contexts in which they operate, and the assumptions they make about the nature of models and their intended use. Despite these differences, there are a number of common concerns that all merge frameworks face. These concerns primarily have to do with the *methodological* requirements of model merging and involve important questions about how to specify the relationships between models, maintain traceability between models and their merges, and capture the evolution of models during merge.

In this position paper, we focus on the question of how to specify the relationships between models. Specification of model relationships is a crucial step in the merge process for characterizing how the contents of different models overlap or interact. In the past several years, we have been studying the model merging problem in different domains. With hindsight, we have identified two key factors concerning the specification of model relationships:

1. Models are often developed by distributed teams. As a result, one can never be entirely sure how the concepts in different models relate to one another. Therefore, it is important to be able to hypothesize and explore alternative ways of interrelating the concepts in a set of models and to configure model merges based on these alternatives.

2. Model merging may be utilized to achieve different goals at different stages of development. For example, in early stages, we may merge a set of models as a way to unify the vocabularies of different stakehold-

*Parts of this work were presented at the posters track of SIGSOFT FSE'06, and an abstract appeared in Software Engineering Notes, V. 31, No. 6, November 2006.

29th International Conference on Software Engineering Workshops(ICSEW'07)
0-7695-2830-9/07 $20.00 © 2007 IEEE

ers. For this purpose, it is convenient to treat models as syntactic objects without rigorous semantics. In later stages, however, we often want to account for the semantics of models and produce merges that preserve certain semantic properties. This means that we may need to apply different merge algorithms to a set of models as they move through different stages of development. These algorithms typically require different types of relationships to be built between models (e.g. syntactic and semantic). Hence, we need the flexibility to define different types of model relationships and to associate each type with certain merge algorithms.

These factors emphasize the need to make explicit all information about the relationships between models, and indeed suggest that model relationships should be treated as *first-class artifacts* in the model merging problem. Such treatment, as first noted by Bernstein et al. [5], will also simplify the implementation of the merge operation.

In light of this observation, we propose a general framework for model merging whereby the relationships between models can be specified explicitly and used for driving the merge process. Figure 1 shows an outline of our framework: Given a set of models, users can define different types of relationships between them and for each type, explore alternative ways of mapping the models to one another. The models and a selected set of relationships are then used to compute a merge by applying an appropriate merge algorithm. The merge is then presented to users for further analysis which may lead to the discovery of new relationships or the invalidation of some of the existing ones. Users may then want to start a new iteration by revising the relationships and executing the subsequent activities.

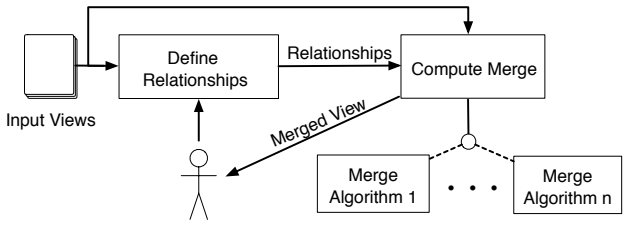


Figure 1. Outline of our framework

We have implemented a proof-of-concept tool, TReMer [1] [18], based on our proposed framework. We describe this tool in the next section.

2 Tool Support

Our tool, TReMer, is an instantiation of the framework outlined in Figure 1 to the domain of state-machine modelling. TReMer currently supports two model merging algorithms, *structural* and *behavioural*, as described below.

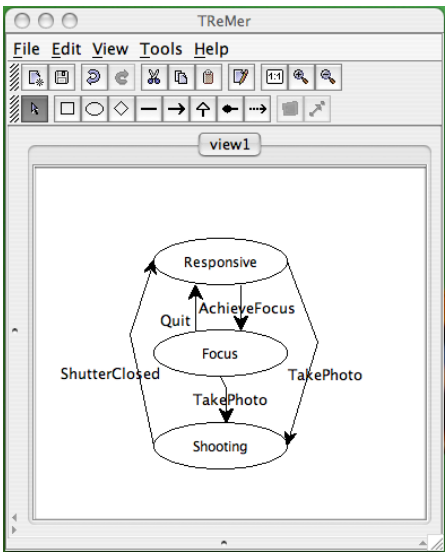
Structural merging is used during early phases of development where the most important goals are: exploring the ontological relationships between the terms used in different models, aligning the conceptual structures of different stakeholders, toleration of inconsistencies and deferral of their resolution to later phases, and producing an overall perspective of the system being developed. Our structural merging algorithm treats state-machines as graphs [16]. Relationships between state-machines are captured by subgraphs, also referred to as *connectors*. Merges are computed based on a category-theoretic concept called *colimit* [3]. Colimit computation is based on syntactic relationships between graphs, without regard to their underlying semantics. Therefore, merges may not preserve the semantic properties of state-machines.

Our second algorithm, behavioural merge, is motivated by the need to preserve the behavioural properties of a set of state-machines in their merges. The desirable behaviours of each state-machine are described using temporal logic formulas. Behavioural merging is more suitable for late stages of development where the goal is to obtain a sound and operational model of the system under development. In behavioural merging, it is assumed that any disagreements between stakeholders have already been resolved, and as such, all remaining discrepancies between the behaviours of the input models are treated as variabilities in the models' intended functionality. These variabilities are represented as conditional behaviours in the merge. Relationships between state-machines are specified by binary relations over their state spaces. The merge is defined as a *common refinement* of a set of state-machines with respect to their relationships [19, 14]. This guarantees the preservation of temporal properties. Merges computed by our behavioural merging algorithm can potentially be used for synthesizing a correct implementation of the final system.

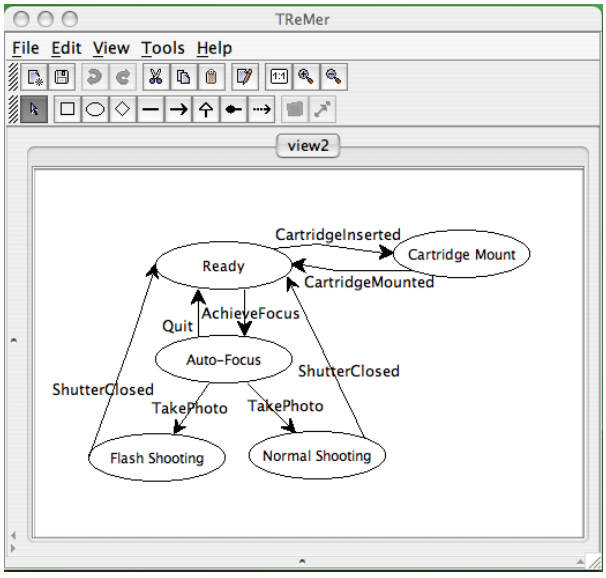
TReMer consists of two main components: (1) A graphical front-end allowing users to visually express their models, specify different relationships between these models, and hypothesize and compute merges, and (2) A library of merge algorithms. These algorithms communicate with the front-end in a uniform way via an abstract interface for the merge operation. The key idea that allows us to have such an interface is the treatment of model relationships as first-class artifacts. For details of our merge algorithms, refer to [16, 17] and [14].

3 Examples of Model Merging

In this section, we illustrate our tool through concrete examples. We use two views on the photo-taking functionality of a camera: Figure 2(a) shows a perspective where the flash and non-flash (normal) shooting modes are not distinguished and the film-cartridge loading procedure is ignored. Further, the perspective has a transition from **Responsive** to



(a) Without Flash



(b) With Flash

Figure 2. Two perspectives on a camera

Shooting meaning that the camera can take a photo even without achieving focus. In the second perspective, Figure 2(b), flash and non-flash shooting are distinguished and cartridge loading is modelled.

3.1 Structural Model Merging

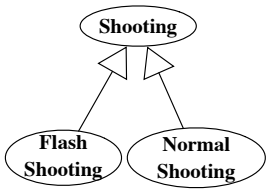
We demonstrate structural merging over the state-machines in Figure 2. In the first step, the user creates a connector model, which includes just the overlaps between Figures 2(a) and 2(b), and specifies how the connector maps onto each of the two models. To describe a mapping between the connector and each model, the connector and the model are shown side-by-side. An element correspondence is established by first clicking on an element of the connector and then on an element of the model. Figure 3 (resp. Figure 4) shows a mapping between the connector and the model in Figure 2(a) (resp. Figure 2(b)). To show the desired correspondences, we have added to the screenshots a set of dotted lines indicating the related elements.

The second step is to compute the merge of the models in Figure 2 with respect to their overlaps as described by the connector. The result is shown in Figure 5.

Our structural merging approach has a number of features, which we briefly discuss below. For more information, consult [17]. Firstly, the approach can handle uncertainties and disagreements in models – for simplicity, we did not illustrate this feature here. Secondly, the approach can be applied for merging more than two models at a time – it works for any number of models with arbitrary interconnections between them. Thirdly, structural merging results in an abstraction if several elements of one model are mapped to a single element of another. For example, **Flash Shooting** and **Normal Shooting** in Figure 2(b) collapsed into a single state in the merge (Figure 5) because both states correspond to **Shooting** in Figure 2(a).

If such abstraction is undesirable, we need to refine our assumptions about the relationships between the models. Our structural merging approach provides the flexibility to express relations other than exact correspondences, e.g. *specialization* (isA), *aggregation* (hasA), and *similarity* (isSimilarTo). These relations may have no particular meaning in the operational semantics of state-machines; nevertheless, they can play an important role in building conceptual mappings between the contents of different models.

In our example, we can declare **Flash Shooting** and **Normal Shooting** in Figure 2(b) to be specializations of **Shooting** in Figure 2(a). To do this, we remove **Flash Shooting** and **Normal Shooting** from the shared connector, and describe the specialization relations in a *helper model* like the one shown at right. We then incorporate this new model into the merge by describing how it overlaps with each of the models in Figures 2(a) and 2(b). That is, we map **Shooting** in the helper model to **Shooting** in Figure 2(a), and similarly, map **Flash Shooting** and **Normal Shooting** to the corresponding states in Figure 2(b).



As illustrated above, helper models offer a mechanism for bridging gaps between models that may be at different levels of abstraction. Similar techniques are employed in [13] and [12] for dealing with conceptual discontinuities between models.

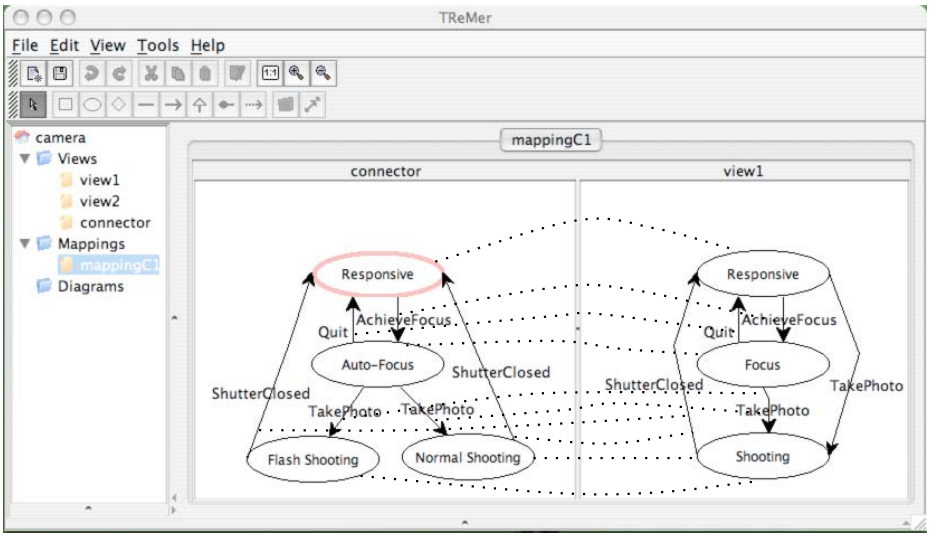


Figure 3. Mapping between the connector and the model in Figure 2(a)

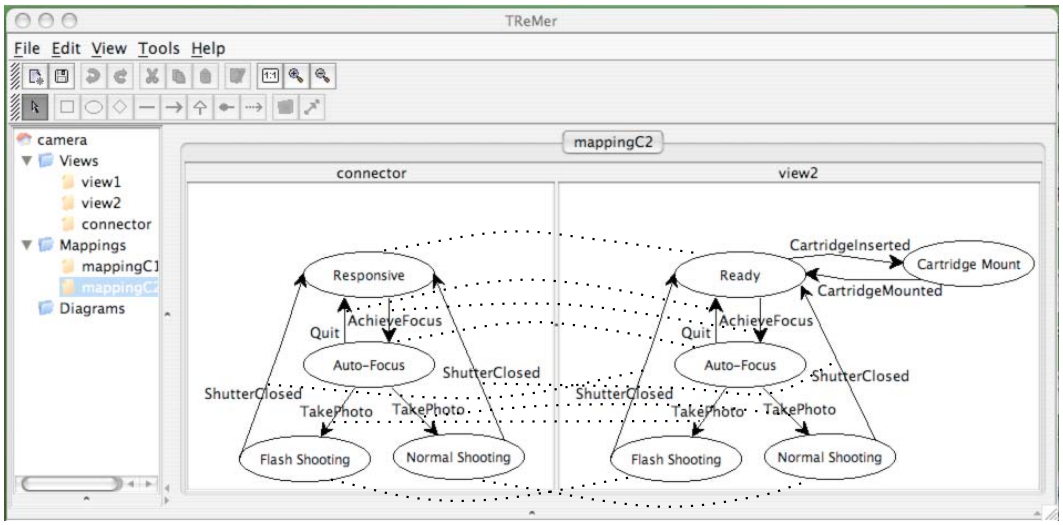


Figure 4. Mapping between the connector and the model in Figure 2(b)

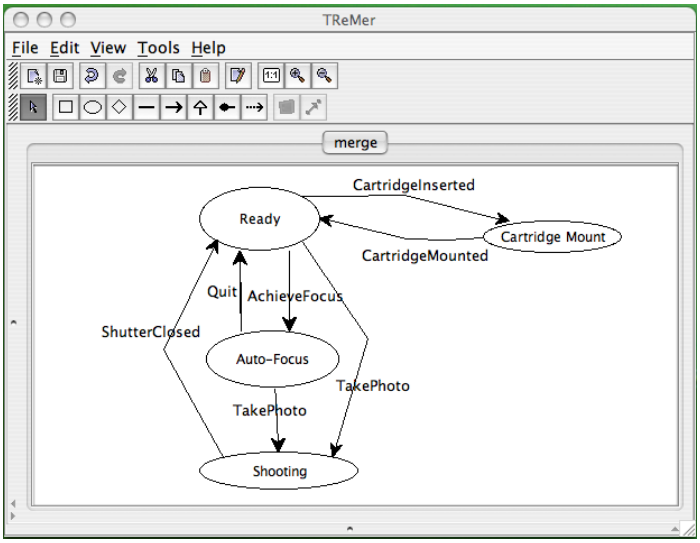


Figure 5. Structural merge of the models in Figure 2

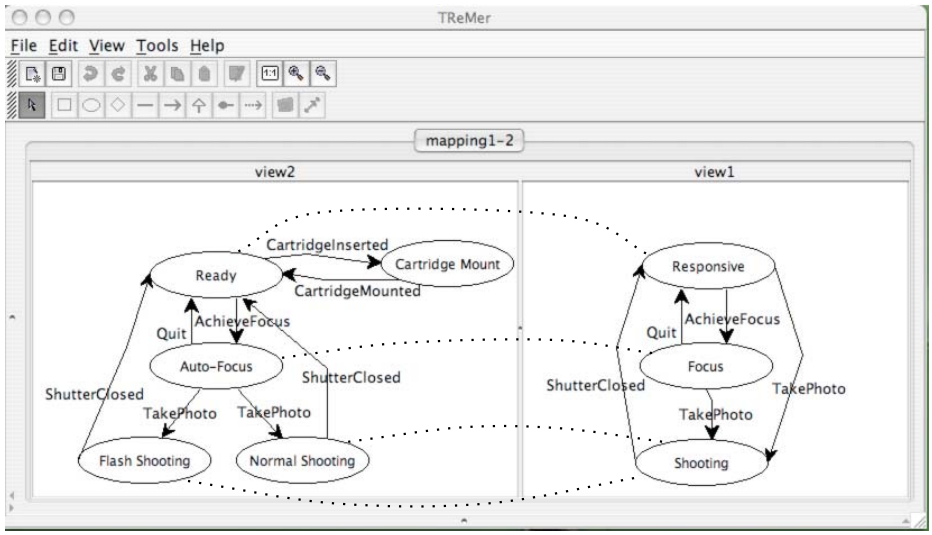


Figure 6. Behavioural mapping between the models in Figure 2

3.2 Behavioural Model Merging

Behavioural merging of the state-machines in Figure 2 proceeds as follows: In the first step, the user defines a mapping between the states of the input models. This is done in a similar manner to the structural approach. However, in contrast to the structural approach, a correspondence between two states is not interpreted as a declaration of them being equal, but rather, as a prescription for combining their behaviours in the merge. Such interpretation of correspondences makes it possible to establish direct mappings between models at different levels of abstraction.

Figure 6 shows a behavioural mapping between the input state-machines. Note that behavioural mappings do not need to relate transitions because, under bisimilarity semantics, parallel transitions with the same label between a pair of states can be replaced by a single transition.

The next step is to compute the merge with respect to the mapping defined above. The result is shown in Figure 7. As can be seen from the figure, non-shared behaviours are guarded by conditions denoting the originating model that exhibits those behaviours. Further, unlike structural merging, behavioural merging does not collapse distinct states. For example, the **Normal Shooting** and **Flash Shooting** states of Figure 2(b) remain intact in the merge (Figure 7) although the two states are mapped to a single state of Figure 2(a). This is necessary for preserving the common behaviours of the input state-machines in their merge.

The merge preserves, in either guarded or unguarded form, *every* behaviour of the input models. For example, the property that "whenever focus is achieved, we can take a picture" holds in both state-machines of Figure 2, and as such, is incorporated as an unguarded behaviour in the merge in Figure 7. However, the property that "we can take

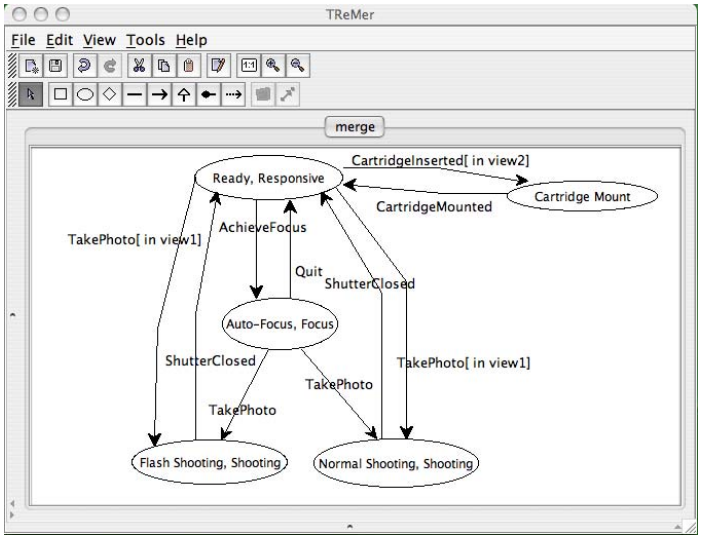


Figure 7. Behavioural merge of the models in Figure 2

a picture without achieving focus", which holds only in the model of Figure 2(a), is represented as a parametrized behaviour in the merge and can occur only when the guard `[in view1]` holds. A change in the mapping between the input models does not cause any behaviours to be added to or removed from the merge, but may make some guarded behaviours unguarded, or vice versa. The use of parameterization for representing behavioural variabilities allows to generate behaviour-preserving merges for models that may even be inconsistent.

Behavioural merging can be used for merging more than two models by iteratively merging a new input model with

the result of a previous merge, with one minor modification: transition guards will range over subsets of input model indices rather than individual model indices. For a fixed set of mappings between the models, the order in which binary merges are applied does not affect the final result.

4 Discussion and Future Work

The models used in this paper for illustrating model merging were quite small, making it relatively easy to identify their relationships by hand. For large models, relationships are often difficult to identify manually. This necessitates the development of appropriate matching techniques for finding correspondences between elements of different models. Matching is a heuristic process; therefore, matching results need to be reviewed by a domain expert and adjusted if necessary. In [14], we have developed a match operator for behavioural models and are now extending our work to structural models. We are also studying the interactions between model merging and matching. Some initial work in this direction has been reported in [6, 14].

Another important aspect of our future work is improving usability of our model merging tool. This allows us to apply our framework to larger case-studies and to conduct an evaluation of its effectiveness. An evaluation of our work will have to provide answers to the following key questions: How much does our framework improve time-to-completion of different model merging tasks? And, how do merges generated by our framework compare with manually-generated merges in terms of quality attributes such as integrity and understandability?

Acknowledgments. We thank Zinovy Diskin and the anonymous reviewers of the MiSE workshop for their insightful comments. Financial support was provided by Bell Canada (through the Bell University Labs), OGS, NSERC, and an IBM Eclipse innovation grant.

References

[1] M. Abi-Antoun, J. Aldrich, N. Nahas, B. Schmerl, and D. Garlan. Differencing and merging of architectural views. In *21st International Conference on Automated Software Engineering (ASE'06)*, pages 47–58, 2006.

[2] M. Alanen and I. Porres. Difference and union of models. In *6th International Conference on The Unified Modeling Language, Modeling Languages and Applications (UML'03)*, pages 2–17, 2003.

[3] M. Barr and C. Wells. *Category Theory for Computing Science*. CRM, Montréal, Canada, 1999.

[4] C. Batini, M. Lenzerini, and S. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.

[5] P. Bernstein, A. Halevy, and R. Pottinger. A vision of management of complex models. *SIGMOD Record*, 29(4):55–63, 2000.

[6] G. Brunet, M. Chechik, S. Easterbrook, S. Nejati, N. Niu, and M. Sabetzadeh. A manifesto for model merging. In *Proceedings of the 1st International Workshop on Global Integrated Model Management*, pages 5–12, 2006.

[7] Z. Diskin, J. Dingel, and H. Liang. Scenario integration via higher-order graphs. Technical Report TR-2006-517, School of Computing, Queen's University, Kingston, Canada, 2006.

[8] S. Easterbrook and B. Nuseibeh. Using viewpoints for inconsistency management. *Software Engineering Journal*, 11(1):31–43, 1996.

[9] A. Egyed. *Heterogeneous View Integration and its Automation*. PhD thesis, University of Southern California, Los Angeles, USA, 2000.

[10] A. Finkelsetin, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: A framework for integrating multiple perspectives in system development. *International Journal of Software Engineering and Knowledge Engineering*, 2(1):31–58, 1992.

[11] Y. Kalfoglou and M. Schorlemmer. Ontology mapping: The state of the art. In *Semantic Interoperability and Integration*, number 04391 in Dagstuhl Seminars, 2005.

[12] J. Madhavan, P. Bernstein, P. Domingos, and A. Halevy. Representing and reasoning about mappings between domain models. In *18th National Conference on Artificial Intelligence*, pages 80–86, 2002.

[13] N. Medvidovic et al. Software model connectors: Bridging models across the software lifecycle. In *13th International Conference on Software Engineering and Knowledge Engineering*, pages 387–396, 2001.

[14] S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, and P. Zave. Matching and merging of statecharts specifications. In *29th International Conference on Software Engineering (ICSE'07)*, 2007. To Appear.

[15] D. Richards. Merging individual conceptual models of requirements. *Requirements Engineering Journal*, 8(4):195–205, 2003.

[16] M. Sabetzadeh and S. Easterbrook. Analysis of inconsistency in graph-based viewpoints: A category-theoretic approach. In *18th International Conference on Automated Software Engineering (ASE'03)*, pages 12–21, 2003.

[17] M. Sabetzadeh and S. Easterbrook. View merging in the presence of incompleteness and inconsistency. *Requirements Engineering Journal*, 11(3):174–193, 2006.

[18] M. Sabetzadeh and S. Nejati. TReMer: A tool for relationship-driven model merging. In *14th International Symposium on Formal Methods (FM'06)*, 2006. Tool Demonstration.

[19] S. Uchitel and M. Chechik. Merging partial behavioural models. In *12th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 43–52, 2004.

[20] J. Whittle and J. Schumann. Generating statechart designs from scenarios. In *22nd International Conference on Software Engineering (ICSE'00)*, pages 314–323, 2000.

[21] A. Zito, Z. Diskin, and J. Dingel. Package merge in uml 2: Practice vs. theory? In *9th International Conference on Model Driven Engineering Languages and Systems (MoDELS'06)*, pages 185–199, 2006.