

# Model Management for Continuously Evolving Systems

Steve Easterbrook  
Department of Computer Science, University of Toronto  
Toronto, Ontario, Canada  
sme@cs.toronto.edu

## 1 Requirements and Systems-of-Systems

Software development today takes place in the context of a complex system-of-systems that includes a broad technological infrastructure along with a wide set of human activities. The technological systems and the human activity systems have a symbiotic relationship - each shapes the other in complex ways, such that neither can be understood in isolation. A recent report from the SEI on Ultra-Large Scale (ULS) Systems accurately characterized the nature of these systems-of- systems: they have no centralized control; experience normal failures and continual evolution of heterogeneous elements; and their requirements are inherently conflicting, diverse and often unknowable. For design purposes, the boundary between people and software disappears - design is as much about shaping the human activities as it is about constructing the software. Although the SEI report focussed on the extreme scale, it is clear that most of these challenges now apply to most software development - the complexity arises from the many different contexts in which any software component will be deployed.

The engineering approaches we use today for software development only work when we take a very narrow view of the requirements, as well-defined sets of features and interfaces, which can be fully specified. This approach helps us to build components that conform (in a narrow sense) to their specifications. But we cannot tell in advance whether they will be any use in any of the many different systems-of-systems in which they may be deployed. Our engineering techniques rapidly break down when we attempt to scale up our design ambitions. The result is a growing gap between expectations and practice in the software industry. We can build very reliable software at the small scale, for tightly constrained problems. But we cannot build reliable software for complex socio-technical domains. Our designs are brittle - they fail in unexpected ways when the context in which they operate changes.

To make progress on these challenges we need to abandon the idea that we can write complete, consistent requirements specifications. Instead, we need to capture the multi-

ple, conflicting requirements for each software component that arise from its different contexts of use. We need to be able to express our partial understandings of the broader systems-of-systems in which our components will be deployed. And we need to be able to reason about the properties, and end-to-end behaviours of these systems, without resolving all the unknowns and inconsistencies in our models. In particular, we need techniques for managing large, evolving collections of fragmentary requirements models, and techniques to reason about the properties of these models and their interactions, even in the presence of inconsistency and incompleteness.

## 2 Model Fusion

Our current research focuses on *model management*, and specifically the challenge of reasoning over multiple, inconsistent models. We assume that requirements models are constructed and manipulated by distributed teams, each working on a partial view of the overall system. Model management involves keeping track of the relationships between the various models, and, from time to time, combining information from several models into a single model. We call the latter process *model fusion*.

A number of factors make model fusion complicated. The modelers may have used different vocabularies, or applied a shared vocabulary inconsistently. Models may overlap, in that they refer to the same concepts in the requirements or design of a system, but the overlapping concepts may be presented differently in each model, and the models may contradict one another. Models may evolve through a number of different versions, so that model fusions may need to be recomputed if the source models are updated.

The problem of model fusion has been studied in many domains: schemata of several independently developed databases [3, 17, 11], static and dynamic UML models [24, 6], web services [10], program variants [8, 15], requirements models [19, 20], and many varieties of reactive systems [9, 23, 4, 7].

The modeling formalisms and the underlying assump-

tions in these approaches differ significantly. For example, most of these approaches require that only consistent models be combined, implying that inconsistent models must be repaired prior to model fusion [14]. However, some approaches tolerate inconsistency, and can represent the inconsistencies explicitly in the resulting fused model (e.g. [4, 20, 15]). In some cases, the goal is to show *differences* between the models rather than to put them together [24].

The approaches disagree on the treatment of information not explicitly present in the model. For example, [3, 11, 7, 4] assume that omitted information is “possible” (e.g., an equation “ $x = 3$ ” does not constrain the value of  $y$ ), whereas most state-machine approaches assume that omitted information signifies prohibited behaviours; [23] takes a hybrid approach.

These approaches further differ in whether they handle heterogeneous modeling notations. Most approaches to model fusion are intended for combining models expressed in the same notation (e.g. fusion of UML Class diagrams with one another [24]) Some approaches solve the problem of model fusion in heterogeneous notations by first translating them into an underlying unified language and only then combine them (e.g. [6]). [7] is a notable exception which allows fusion of scenarios and state machines directly.

Finally, the approaches range in the level of sophistication in matching and identifying similar information. In software engineering, most approaches assume that entities are the same when they have the same name or id (or are derivatives of entities with the same name or id); others, e.g., [19], support the use of an explicit ontology, or a thesaurus, relating entities from each model. In the database work, schema integration approaches implement sophisticated matching algorithms [18].

We believe that distributed model management depends crucially on being able to put together models coming from different sources. To that end, we are developing a framework for comparing different approaches to model fusion, and we are developing a taxonomy of model management operators:

- *fuse* – which takes two or more models along with a specified relationship between them, and combines them into a single model;
- *match* – which takes two or more models and identifies candidate relationships between them by comparing their structure, semantics and/or terminology;
- *diff* – which takes two models and computes the edit difference between them as a number of steps needed to transform one into the other;
- *split* – which takes a model and partitions it into two

or more separate models along with a relationship between them (the inverse to *fuse*);

- *slice* – which takes a model and a projection criterion, and extracts a partial view (projection, aspect, ) that satisfies the given criterion;
- *patch* – which takes a model and a transformation and applies the transformation to the model;
- *propagate* – which takes a series of edit actions (i.e. a transformation) that have been performed on one model, and applies them to another model, according to a specific relationship between the two models.

Our work has led us to a reconsideration of modeling semantics based on how well they support distributed modeling. The intended relationship between two partial models may be at odds with the usual semantics of the modeling formalism. In particular few modeling languages address the notion of missing information (partiality). Notable exceptions for state modeling are approaches based on Live Sequence Charts [7], MTSs [9, 23] and partial Kripke structures [12]).

Finally, this work depends on a more explicit specification of the relationships between models during distributed model development. Some frameworks for model management make use of model versioning information to capture some aspects of the relationships between models in the same version tree. The relationships we have identified go beyond versioning, to include all shared information in multiple models. In our future work, we intend to devote more attention to the question of how to elicit and represent such relationships between models.

### 3 Current Research Goals

Our current research addresses the following goals:

- Identifying and characterizing the set of operations performed in model management tasks, and to characterize the different instances of these operations for different types of model.
- Developing a standard approach for representing connectors between partial and inconsistent models, and for expressing the (desired) properties of sets of inter-related models.
- Designing and implementing a flexible framework (based in Eclipse) that unifies model management operations. The framework will enable analysts to create, view and manipulate models and their connectors, and provide support for deciding how and when to apply the various operators needed for model-driven development.

- Instantiating this framework for several common types of model used in model-based development, including both structural models (e.g. class diagrams) and behavioural models (e.g. statecharts).

Our progress so far is as follows. We have developed the taxonomy of model management operators, and characterized the properties of “ideal” forms of these operators in algebraic terms [2]. We have explored the use of techniques from psychology, specifically, *personal construct theory*, to identify terminological interference between stakeholders views [16]. We have developed a general framework for model fusion based on identification of structural overlaps between models [21]. We have developed tools for matching and fusing hierarchical statechart diagrams [13]. We have identified a constraint language and developed a tool for reasoning over fused models [22]. In addition we have also conducted some preliminary empirical studies on the ideas underlying this work, particularly into the question of how people actually use models to understand a problem situation [5, 1].

## References

- [1] J. Aranda, N. Ernst, J. Horkoff, and S. M. Easterbrook. “A Framework for Empirical Evaluation of Model Comprehensibility”. In *International Workshop on Modeling in Software Engineering (MiSE-07) at the 29th International Conference on Software Engineering (ICSE’07)*, 2007.
- [2] G. Brunet, M. Chechik, S. M. Easterbrook, S. Nejati, N. Niu, and M. Sabetzadeh. “A Manifesto for Model Merging”. In *Workshop on Global Integrated Model Management (GaMMa’06) at the 28th International Conference on Software Engineering*, 2006.
- [3] P. Buneman, S. Davidson, and A. Kosky. Theoretical aspects of schema merging. In *Proceedings of the 3rd International Conference on Extending Database Technology*, pages 152–167, 1992.
- [4] S. Easterbrook and M. Chechik. “A Framework for Multi-Valued Reasoning over Inconsistent Viewpoints”. In *Proceedings of International Conference on Software Engineering (ICSE’01)*, pages 411–420, 2001.
- [5] S. M. Easterbrook, E. Yu, J. Aranda, Y. Fan, J. Horkoff, M. Leica, and R. A. Qadir. “Do Viewpoints Lead to Better Conceptual Models? An Exploratory Case Study”. In *Proc. 13th IEEE International Requirements Engineering Conference (RE05)*, 2005.
- [6] A. Egyed and N. Medvidovic. “A Formal Approach to Heterogeneous Software Modeling”. In *Proceedings of Formal Aspects of Software Engineering (FASE’00)*, pages 178–192, 2000.
- [7] D. Harel, H. Kugler, and A. Pnueli. “Synthesis Revisited: Generating Statechart Models from Scenario-Based Requirements.”. In *Formal Methods in Software and Systems Modeling*, pages 309–324, 2005.
- [8] S. Horwitz, J. Prins, and T. Reps. “Integrating Noninterfering Versions of Programs.”. *ACM Transactions on Programming Languages and Systems*, 11(3):345–387, 1989.
- [9] K. Larsen and L. Xinxin. “Equation Solving Using Modal Transition Systems”. In *Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science (LICS’90)*, pages 108–117, 1990.
- [10] N. Liu, J. C. Grundy, and J. G. Hosking. “A Visual Language and Environment for Composing Web Services”. In *Proceedings of Automated Software Engineering (ASE’05)*, pages 321–324, 2005.
- [11] S. Melnik, E. Rahm, and P. Bernstein. “Rondo: a Programming Platform for Generic Model Management”. In *SIGMOD Conference*, pages 193–204, 2003.
- [12] 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.
- [13] S. Nejati, M. Sabetzadeh, M. Chechik, S. M. Easterbrook, and P. Zave. “Matching and Merging of Statecharts Specifications”. In *Proc. of the 29th International Conference on Software Engineering (ICSE’07)*, 2007.
- [14] C. Nentwich, W. Emmerich, and A. Finkelstein. “Consistency Management with Repair Actions”. In *Proc. of the 25 Int. Conference on Software Engineering (ICSE’03)*, 2003.
- [15] N. Niu, S. Easterbrook, and M. Sabetzadeh. “A Category-Theoretic Approach to Syntactic Software Merging”. In *21st IEEE International Conference on Software Maintenance (ICSM’05)*, pages 197–206, 2005.
- [16] N. Niu and S. M. Easterbrook. “So, You Think You Know Others’ Goals? A Repertory Grid Study”. *IEEE Software*, 24(2):53 – 61, 2007.
- [17] R. Pottinger and P. Bernstein. “Merging Models Based on Given Correspondences”. In *Proceedings of 29th*

*International Conference on Very Large Data Bases (VLDB'03)*, pages 862–873, 2003.

- [18] E. Rahm and P. Bernstein. “A Survey of Approaches to Automatic Schema Matching”. *The VLDB Journal*, 10(4):334–350, 2001.
- [19] D. Richards. “Merging Individual Conceptual Models of Requirements”. *Requirements Engineering*, 8(4):195–205, 2003.
- [20] M. Sabetzadeh and S. Easterbrook. “An Algebraic Framework for Merging Incomplete and Inconsistent Views”. In *13th IEEE International Requirements Engineering Conference*, September 2005.
- [21] M. Sabetzadeh and S. M. Easterbrook. “View Merging in the presence of incompleteness and inconsistency”. *Requirements Engineering Journal*, 11:174 – 193, 2006.
- [22] M. Sabetzadeh, S. Nejati, S. Liaskos, S. Easterbrook, and M. Chechik. “Consistency Checking of Conceptual Models via Model Merging”, 2007. Submitted for publication.
- [23] 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.
- [24] Z. Xing and E. Stroulia. “UMLDiff: An Algorithm for Object-Oriented Design Differencing”. In *Proceedings of 20th IEEE International Conference on Automated Software Engineering (ASE'05)*, pages 54–65, 2005.