

# Formal Support for Merging and Negotiation

Shiva Nejati  
Dept. of Computer Science, University of Toronto  
Toronto, Ontario, Canada  
shiva@cs.toronto.edu

## ABSTRACT

Model merging is an important activity in software development. We often need to integrate a set of models coming from different sources so as to create a unified model encompassing all the given models. Inconsistencies between models can make model merging significantly more complex. To deal with inconsistencies efficiently, a systematic negotiation process is needed. This paper outlines a formal approach to merging and negotiation over behavioural models and presents the results achieved so far.

**Categories and Subject Descriptors:** D.2.1 [Software Engineering]: Requirements/Specifications - *methodologies*; D.2.4 [Software Engineering]: Software/Program Verification - *model checking*.

**General Terms:** Design, Theory, Verification.

**Keywords:** Model Merging, Refinement, Model Checking, Inconsistency Detection, Negotiation, 3-Valued Logic.

## 1. RESEARCH QUESTION

Almost every kind of software development periodically needs to merge models. For example, during requirements analysis, different stakeholders with different viewpoints [26] describe different, yet overlapping aspects [3] of the same systems. How should these partial models be put together? Alternatively, consider combining behavioural models of component instances of the same type. Typically, several instances of the same component may appear in a given scenario, e.g., several instances of a client component that concurrently access a server [30]. Standard approaches to synthesis produce a separate behavioural model for each client instance (e.g., [31, 21]). It is reasonable to integrate all models of all client instances into a single model for the client component type because all clients should share the same characteristics.

The problem gets even more pressing when we are dealing with distributed software development [5], when teams in different locations independently modify a common model, and then attempt to put their modifications together.

In the context of model elaboration, composition of two (partial) descriptions of the *same* component to obtain a more elaborate ver-

sion of the original partial description has been called *merge* [30]. Effective merging supports collaboration and cooperation in the process of specifying software and helps manage the complexities of this process. Unfortunately, merging can combine models only if there are no disagreements between the stakeholders. Otherwise, this composition requires *negotiation*.

Merging and negotiation go hand in hand. In order to effectively support software development, we need to be able to merge different versions of software models and support possible conflict resolution. We believe the process of merging and negotiation of software models should be supported by a formal framework.

### 1.1 Research Objective

The main objective of our research is to propose, develop and study a framework for merging and negotiation over behavioural software models. Behavioural models are typically divided into declarative specifications, such as Alloy [19], and operational specifications, expressed in some form of state-machines. There are different kinds of state-machines, e.g., state-based/event-based, flat/hierarchical, or etc. State-machines are widely used in requirements modelling [12, 13, 14] either directly, or via translation from higher-level modeling languages.

A framework for merging and negotiation should be able to merge models, when they are consistent, and otherwise support negotiation by helping users discover their disagreements, allow them to trace through their decisions and understand different alternatives for resolving conflicts.

Given two models, the framework should automatically determine whether the models can be merged and compute the merge if it exists. Otherwise, it should support the negotiation process, helping users identify their disagreements and prioritize them. Further, it should provide automated support for computing proposals to bring users closer to resolving their conflicts, allowing users to do “what if” exploration and choose the most suitable alternatives. We also want to keep a history of decisions that have been made, allowing users to study the results, and, if necessary, undo the decisions.

The most significant factors that we are taking into account to make our framework as effective as possible are: improving the readability and the precision of merges and resulting models; providing users with measures on how inconsistent their models are; helping users manage negotiation when there are too many inconsistencies; and ensuring that during conflict resolution, only a minimal number of changes are imposed onto the original models.

Furthermore, to make the framework applicable to a wide range of software models, we need to support the merging of hierarchical structures as well as models described in different formalisms and at different levels of abstraction.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ASE'05, November 7–11, 2005, Long Beach, California, USA.

Copyright 2005 ACM 1-58113-993-4/05/0011 ...\$5.00.

## 2. RELATED WORK

Formal support for model-merging has been addressed by several researchers. For example, merging is just a conjunction of the corresponding theories in declarative specifications [19]. Uchitel and Chechik [30] define merging for consistent partial labelled transition systems. Like ours, their notion of merge is based on common refinement. However, while able to detect inconsistencies, [30] does not consider the problem of negotiation and conflict resolution.

A number of approaches to inconsistency management have been studied in the context of viewpoint-based modelling [26]. Some of this work, e.g., [9, 25], detects inconsistencies by using first-order logic rules and does not consider merging as a means of model exploration and inconsistency detection. Other researchers [18, 8, 28] propose ways of merging viewpoint models. Huth and Pradhan [18] define the merge as the common refinement of partial state transition systems. They enforce consistency across inconsistent viewpoints by using a dominance ordering on owners of the viewpoints. [8, 28] allow the merge of *inconsistent* viewpoints using multi-valued logic. Their goal is to tolerate disagreement while still enabling reasoning. In [8], states are merged only if they have the same label. The merge in [28] is based on the structural mapping of graph morphisms with the emphasis on preserving structure rather than behavior. Non-classical logics have been used for reasoning with inconsistency by others, e.g., in [15]. Unlike this work, our goal is to merge only consistent models, and we allow users to explore agreement models, which are consistent but incomplete, to determine those inconsistencies that need to be resolved.

In [6], requirements evolution is supported by an iterative process that is similar to our exploration and resolution phases; however, the implementation of the resolution phase is left open, conjecturing that machine learning techniques may be suitable for it. The work in [10] extends [6] by using multi-valued logic to address incompleteness and partiality of requirements specifications. However, [10] does not give the problem a formal treatment, illustrating the process by an example instead.

We are not aware of other formal logic-based approaches to negotiation. The existing work, e.g., [4, 7, 1], is based on dialectic reasoning. In these approaches, requirements are treated as informal generic entities, and the focus is on formalizing the relationships and interactions between them. Even though both functional and non-functional requirements can be handled using these approaches, correctness and completeness become subjective. In contrast, we only consider behavioural requirements, but their formality allows us to perform tool-supported inconsistency resolution while tolerating the less critical inconsistencies.

## 3. PROPOSED SOLUTION

We have recently designed a formal framework for supporting merging and negotiation over state machines [24]. In this section, we briefly discuss our framework.

### 3.1 Assumptions

We concentrate on merging and negotiation over state-based transition systems. We only consider flat models where their entities are specified at the same level of abstraction and are named consistently in each model, i.e., we assume *vocabulary consistency*. Note that stakeholders can use different sets of vocabularies, but they should be consistent with each other. We often assume the existence of certain global properties for ensuring that state-machines behave as expected. Model invariants, use-cases, and scenarios are examples of such properties. Global properties may assist users in

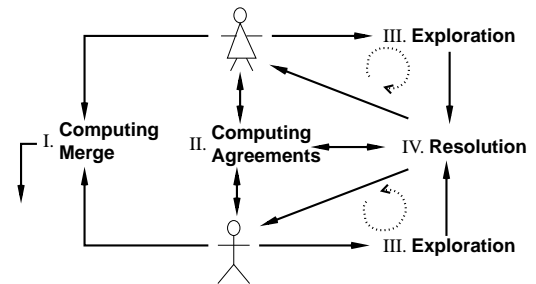


Figure 1: Negotiation framework.

the process of negotiation and in prioritizing inconsistencies. Nevertheless, our framework can work without global properties too.

### 3.2 Negotiation Framework

The overall picture of the framework is illustrated in Fig. 1. Given two state-machines, we determine whether the models can be merged, and if so, the merge is computed in the **Computing Merge** phase (Section 3.2.1). In situations where models are inconsistent and their merge does not exist, we start the negotiation process that consists of three phases: **Computing Agreements**, **Exploration**, and **Resolution**. We first build a model that reflects agreements between the original models (**Computing Agreements**, Section 3.2.2). Effectively, we label all points of disagreements with *maybe*. Next, we project the result back onto the original models, allowing users to explore the result (**Exploration**, Section 3.2.3). Using global properties or their intuition, users pick a list of items that they care about most. These become input to the **Resolution** phase (Section 3.2.4) which attempts to handle these disagreements by building consistent proposals. Finally, users pick a proposal that suits their goals.

#### 3.2.1 Computing Merge

We use state-machines with *3-valued* transitions and propositions as our modelling formalism. 3-valued state-machines are desirable because they allow us to explicitly capture the incompleteness involved in requirements models. 3-valued logic [20] extends classical logic with an additional truth value, denoted *maybe* (*m*) and interpreted as *unknown*.

The intuition we wish to capture by merge is that of combining partial knowledge coming from individual models while preserving all of their agreements. The notion of *refinement* [17, 22] over 3-valued models underlies this intuition. Refinement over 3-valued state-machines captures the notion of elaboration of an incomplete model into a more complete one, in which the value of some *m* transitions or propositions is changed to true or false. Refinement can be seen as a “more complete than” relationship between 3-valued state-machines.

We define the merge of two state machines  $M_1$  and  $M_2$  to be their *common refinement*. A 3-valued model  $M_3$  is a common refinement of  $M_1$  and  $M_2$  iff  $M_3$  refines both  $M_1$  and  $M_2$ . Basing the notion of merge on a common refinement is standard and has been considered by several researchers [30, 18, 16]. A common refinement preserves common behaviours and temporal properties of the original models [17], and hence, is a reasonable candidate for the merge.

Note that a common refinement for two 3-valued models might not exist at all. In this case, models are *inconsistent* and we start the negotiation process.

#### 3.2.2 Computing Agreements

When models contain inconsistencies, we need to help users identify, understand and resolve them. To this end, it is helpful

to construct a model that preserves the agreements and highlights the disagreements between the models. We refer to it as the *agreement* model and define it as a *common abstraction* between the two models. A 3-valued model  $M_3$  is a common abstraction of  $M_1$  and  $M_2$  iff both  $M_1$  and  $M_2$  refine  $M_3$ . For given models  $M_1$  and  $M_2$ , there are several common abstractions. We seek a common abstraction of  $M_1$  and  $M_2$  that preserves the most common properties of  $M_1$  and  $M_2$  and represents the maximum agreements between them. To this end, we need to find a relation  $\rho$  over the states of  $M_1$  and  $M_2$  that reflects the most similarities between these two models.

Finding the best  $\rho$  is essentially an optimization problem with complexity exponential in the number of the states of  $M_1$  and  $M_2$ . While this may still be feasible for relatively small models, we can use various heuristics for large models instead. Further investigation into effective computations of the best  $\rho$  is left for the future work.

The agreement model captures the common behaviours of  $M_1$  and  $M_2$  and leaves all points of inconsistencies as *m*. We *project* the agreement model back to the context of the original models to allow users to study the agreements and the disagreements between their models in the context of their original models. We denote the projections corresponding to  $M_1$  and  $M_2$  by  $\hat{M}_1$  and  $\hat{M}_2$  respectively. Since the disagreements between  $M_1$  and  $M_2$  are converted to *m* behaviours in the projections  $\hat{M}_1$  and  $\hat{M}_2$ , it can be proven that  $\hat{M}_1$  and  $\hat{M}_2$  are consistent w.r.t.  $\rho$ , and further,  $\hat{M}_1$  and  $M_2$  as well as  $M_1$  and  $\hat{M}_2$  are pair-wise consistent w.r.t.  $\rho$ .

### 3.2.3 Exploration

The **Computing Agreements** phase produces consistent but incomplete projections  $\hat{M}_1$  and  $\hat{M}_2$ . The missing information, represented by *m* in the two models, effectively comes from “backing down” from all disagreements between the original models. Clearly,  $\hat{M}_1$  and  $\hat{M}_2$  can be merged, but the result leaves a number of properties, perhaps the ones which are vitally important to the stakeholders, inconclusive. On the other extreme, we can attempt to reach agreement over every *m* item (i.e., a variable or a transition). However, the number of proposals for resolving inconsistencies can grow exponentially with the number of items; thus, the smaller the list of negotiation items, the easier it is for the stakeholders to reach agreement.

The goal of the **Exploration** phase is to choose the truly important items, which must be negotiated, from the overall list. We call this a *priority list (PL)*. In order to build the PLs, users can either informally inspect the projections or use various analysis tools such as model-checkers, simulators, debuggers etc. Specifically, if a set of global properties is available, users may want to see the impact of rolling back the disagreements on these. In our framework, we assume that global properties are expressed in temporal logic, so any 3-valued model-checker, e.g.,  $\chi$ Chek [2], can be used for this analysis.  $\chi$ Chek can return a counterexample explaining why the property evaluates to *m*. We can extract the list of *m* variables and transitions from the counterexample that is produced by  $\chi$ Chek and report it to the user as the PL. We also report the size of the PL to the user. This gives her an early indication about the feasibility of achieving agreements during the resolution step.

### 3.2.4 Resolution

The goal of the **Resolution** phase is to compute alternatives for resolving the most important inconsistencies identified during the **Exploration** phase, while making minimal possible modifications to the original models. The **Resolution** phase is carried out by a resolution algorithm. The algorithm receives as input the projec-

tion models  $\hat{M}_1$  and  $\hat{M}_2$ , a consistency relation  $\rho$  between  $\hat{M}_1$  and  $\hat{M}_2$ , and the priority lists of inconsistencies. It computes a list of model pairs  $(\hat{M}_1', \hat{M}_2')$ , called *proposals*, which resolve these inconsistencies. The goal of the algorithm is to change the value of every *m* proposition or transition in the PL to either *t* or *f*, resulting in models  $\hat{M}_1'$  and  $\hat{M}_2'$  which remain consistent with respect to  $\rho$ . Sometimes when  $\rho$  is too restrictive, the resolution algorithm may conflict with  $\rho$  and fails to resolve an *m* item. In this case, the tuples in  $\rho$  that cause the conflict are reported back to **Computing Agreements** phase (see Fig. 1), which attempts to find a better  $\rho$ , and then the resolution process is repeated again.

Proposals generated by the resolution algorithm are consistent w.r.t.  $\rho$ , just like the corresponding projections have been, but they refine these projections, deeming more properties conclusive.

Unfortunately, the number of generated proposals can be very large, i.e., exponential in the size of the PL, because the resolution of each item can potentially lead to two proposals. Usually, items in the priority lists depend on each other. This effectively reduces the overall number of generated proposals. Moreover, in situations where the number of proposals is very large, we envision that users will partition their PLs, so that the negotiations can concentrate only on the chosen items. Additional iterations of the framework would be required to resolve the remaining items. Effectively, this enables compositional negotiation. We are still working on a methodology to help users partition their PLs.

## 4. CONTRIBUTIONS AND PRELIMINARY RESULTS

In this section, we discuss the most significant features of our approach to merging and negotiation as well as the implementation and preliminary evaluation of our framework.

We define a merge of two consistent models as their common refinement. We already discussed the advantages of defining a merge as a common refinement in Section 3.2.1, and further in [24], we showed how a common refinement can be computed efficiently. In our framework, negotiation is carried out through a three-phase process: **Computing Agreements**, **Exploration**, and **Resolution**. In the **Computing Agreements** phase, we compute an agreement model and the projections of the agreement model onto the original models. The agreement model allows stakeholders to identify the commonalities and highlight the differences; and the projections allow them to study these commonalities and differences in the context of their original models. Projections are consistent abstractions of the original inconsistent models and show to what degree each model needs to be abstracted so as to become consistent with the other one.

The **Exploration** phase helps users to prioritize the inconsistencies and identify the most critical ones. The size of priority lists (PLs) provides users with a measure on how inconsistent their models are. Finally, the **Resolution** phase allows users to browse different alternatives for resolving inconsistencies and to do “what if” exploration to pick an alternative that suits the global properties and their intuition.

We have prototyped a proof of concept implementation of the merging and negotiation framework discussed in Section 3.1. The implementation can merge 3-valued state-machines if they are consistent. If models are inconsistent, it computes an agreement model, and, using a multi-valued model-checker  $\chi$ Chek [2], allows stakeholders to explore its projections and build their priority lists. Once proposals are built, users can choose their favourite, apply suggested resolutions to their *original* models, and attempt to merge them again.

To evaluate our framework, we attempted to merge two inconsistent descriptions of behaviour of an authentication system, adopted from [29]. We used a few heuristics to improve the precision and the readability of the agreement and projection models in this case-study. These heuristics can facilitate the scalability of the framework, in general. Further details about the heuristics and the case-study are available in [23].

## 5. FUTURE WORK

Several research problems need to be solved to achieve an effective merging and negotiation framework:

- Extending the framework to work with richer behavioural models, e.g. hierarchical state-charts.
- Studying the relationships between state-based and event-based models and the possibility of adapting the results of this work to event-based models.
- Adding support for merging models that are described at different levels of abstraction.
- Evaluating the framework through more case-studies and investigating further heuristics to improve the performance and the scalability of our framework.
- Devising an efficient algorithm for computing mappings that capture the maximal similarities between inconsistent models.
- Developing methods for managing negotiation when there are too many inconsistencies. Currently, we allow users to prioritize and partition their PLs to reduce the complexity of the **Resolution** phase.
- Providing users with measures to represent the amount of inconsistencies between their models, e.g., in the current work, we use the size of PLs for this purpose.
- Changing the resolution algorithm to produce more interesting proposals.

## 6. ACKNOWLEDGMENTS

I thank my advisor, Marsha Chechik, for helpful discussions and her insightful comments.

## 7. REFERENCES

- [1] B. Boehm and H. In. "Identifying Quality-Requirement Conflicts". *IEEE Software*, 13(2), 1996.
- [2] M. Chechik, B. Devereux, and A. Gurfinkel. "XChek: A Multi-Valued Model-Checker". In *CAV*, 2002.
- [3] S. Clarke and R. J. Walker. "Composition Patterns: An Approach to Designing Reusable Aspects". In *ICSE*, 2001.
- [4] J. Conklin and M. Begeman. "gIBIS: A Hypertext Tool for Exploratory Policy Discussion". *Trans. on Info. Sys.*, 4(6), 1988.
- [5] D. Damian, A. Eberlein, M. Shaw, and B. Gaines. "An Exploratory Study of Facilitation in Distributed Requirements Engineering". *REJ*, 8(1), 2003.
- [6] A. d'Avila Garcez, A. Russo, B. Nuseibeh, and J. Kramer. "An Analysis-Revision Cycle to Evolve Requirements Specifications". In *ASE*, 2001.
- [7] S. Easterbrook. "Resolving Conflicts Between Domain Descriptions with Computer-Supported Negotiation". In *Workshop on Knowledge Acquisition for Knowledge Based Systems*, 1990.
- [8] S. Easterbrook and M. Chechik. "A Framework for Multi-Valued Reasoning over Inconsistent Viewpoints". In *ICSE*, 2001.
- [9] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. "Inconsistency Handling in Multi-Perspective Specifications". *IEEE TSE*, 20(8), 1994.
- [10] J. Garcia-Duque, J. Pazos-Arias, and B. Barragans-Martinez. "An Analysis-Revision Cycle to Evolve Requirements Specifications by Using the SCTL-MUS Methodology". In *ICRE*, 2002.
- [11] T. Gruber. "The Role of Common Ontology in Achieving Sharable, Reusable Knowledge Bases.". In *Principles of Knowledge Representation and Reasoning*, 1991.
- [12] D. Harel. "StateCharts: A Visual Formalism for Complex Systems". *Science of Computer Programming*, 8, 1987.
- [13] M. Heimdahl and N. Leveson. "Completeness and Consistency in Hierarchical State-Based Requirements". *IEEE TSE*, 22(6), 1996.
- [14] C. Heitmeyer, A. Bull, C. Gasarch, and B. Labaw. "SCR\*: A Toolset for Specifying and Analyzing Requirements". In *COMPASS*, 1995.
- [15] A. Hunter and B. Nuseibeh. "Managing Inconsistent Specifications: Reasoning, Analysis and Action". *ACM TOSEM*, 7(4), 1998.
- [16] A. Hussain and M. Huth. "On Model Checking Multiple Hybrid Views". In *Inter. Symp. on Leveraging Apps. of FMs*, 2004.
- [17] M. Huth, R. Jagadeesan, and D. A. Schmidt. "Modal Transition Systems: A Foundation for Three-Valued Program Analysis". In *ESOP*, 2001.
- [18] M. Huth and S. Pradhan. "Model-Checking View-Based Partial Specifications". *Electr. Notes Theor. Comp. Sci.*, 45, 2001.
- [19] D. Jackson. "Alloy: A Lightweight Object Modelling Notation". *ACM TOSEM*, 11(2), 2002.
- [20] S. C. Kleene. *Introduction to Metamathematics*. New York: Van Nostrand, 1952.
- [21] I. Krueger, R. Grosu, P. Scholz, and M. Broy. "From MSCs to Statecharts". In *Conf. Distributed and Parallel Embedded Systems*, 1999.
- [22] K. Larsen and B. Thomsen. "A Modal Process Logic". In *LICS*, 1988.
- [23] S. Nejadi. "Negotiation for a B2B E-Commerce Site". <http://www.cs.toronto.edu/~shiva/examples>.
- [24] S. Nejadi and M. Chechik. "Lets Agree to Disagree". *CSRG Tech. Report 530*, Univ. of Toronto, 2005.
- [25] C. Nentwich, W. Emmerich, A. Finkelstein, and E. Ellmer. "Flexible Consistency Checking". *ACM TOSEM*, 12(1), 2003.
- [26] B. Nuseibeh, J. Kramer, and A. Finkelstein. "Framework for Expressing the Relationship Between Multiple Views in Requirements Specifications". *IEEE TSE*, 20(10), 1994.
- [27] E. Rahm and P. A. Bernstein. "A Survey of Approaches to Automatic Schema Matching". *VLDB Journal*, 10(4), 2001.
- [28] M. Sabetzadeh and S. Easterbrook. "Analysis of Inconsistency in Graph-Based Viewpoints". In *ASE*, 2003.
- [29] S. Uchitel and M. Chechik. "Merging MTSs for a B2B E-Commerce Site". <http://www.doc.ic.ac.uk/~su2/merge/examples>.
- [30] S. Uchitel and M. Chechik. "Merging Partial Behavioural Models". In *FSE*, 2004.
- [31] S. Uchitel, J. Kramer, and J. Magee. "Synthesis of Behavioural Models from Scenarios". *IEEE TSE*, 29(2), 2003.