Let's Agree to Disagree

Shiva Nejati Marsha Chechik
Department of Computer Science
University of Toronto
40 St. George Street
Toronto, Ontario, Canada M5S 2E4
Email: {shiva,chechik}@cs.toronto.edu

ABSTRACT

Almost every kind of software development periodically needs to merge models. Perhaps they come from different stakeholders during the requirements analysis phase, or perhaps they are modifications of the same model done independently by several groups of people. Sometimes these models are consistent and can be merged. Sometimes they are not, and negotiation between the stakeholders is needed in order to resolve inconsistencies. While various methods support merging, we need formal approaches that help stakeholders negotiate. In this paper, we present a formal framework for merging and conflict resolution. It facilitates automatic merging of consistent models, enables users to visualize and explore potential disagreements and identify their priorities, and suggests ways to resolve the priority items.

Categories and Subject Descriptors: D.2.1 [Software Engineering]: Requirements/Specifications; D.2.4 Software/Program Verification.

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

1. INTRODUCTION

Almost every kind of software development periodically needs to merge models. For example, during requirements analysis, different stakeholders with different viewpoints [35] describe different, yet overlapping aspects [7] 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 [39]. Standard approaches to synthesis produce a separate behavioural model for each client instance (e.g., [40, 30]). 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 [10], when teams in different locations independently modify a common model, and then attempt to put their modifications together.

In this paper, we concentrate on merging behavioural models of software. There are several ways to express such models; these are typically divided into declarative specifications, such as Alloy [27], and operational specifications, expressed in some form of state-machines. State-machines are widely used in requirements modeling [20, 21, 22] either directly, or via translation from higher-level modeling languages.

In the context of model elaboration, composition of two (partial) descriptions of the *same* component to obtain a more elaborate version of the original partial description has been called *merge* [39]. 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*. In order to support state-machine-based development, we need to be able to merge different versions of state-machines as well as support possible conflict resolution.

Merging and negotiation go hand in hand, and we believe that this process should be supported by a formal framework. Such a framework should merge models, if they are consistent, and otherwise support negotiation by helping users discover their disagreements, allow them to trace through their decisions and understand proposals with the goal of resolving conflicts. This paper presents such a framework in the context of state-machine models. We assume that each entity in our models is specified at the same level of abstraction and is named consistently in each model, i.e., we assume *vocabulary consistency*.

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 [27]. Uchitel and Chechik [39] define merging for consistent partial labelled transition systems, and Huth and Pradhan [26] merge partial view-based specifications where a dominance ordering is used to eliminate the potential inconsistencies. Different aspects of negotiation have been addressed in software engineering literature. For example, [2] describes negotiation over non-functional and application-independent goals, such as the trade-off between assurance and performance or cost/schedule. Damian et. al. [10] consider social and political aspects of negotiation, and [9, 12] take a dialectic reasoning approach to negotiation. Several researchers [13, 23] proposed ways to do formal reasoning with inconsistency; however, we are not aware of formal support for *negotiation* over inconsistent behavioural models.

Given two models, our framework automatically determines whether the models can be merged, and if so, computes the merge. Otherwise, it supports the negotiation process, helping users identify

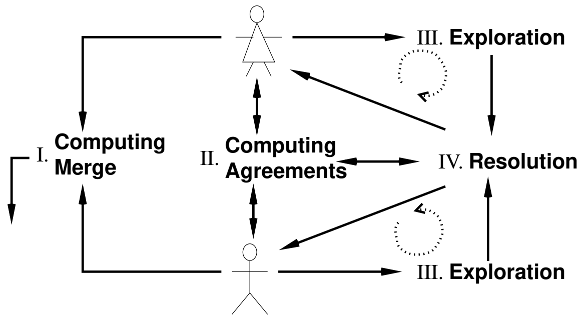

Figure 1: Negotiation framework.

their disagreements and prioritize them. Further, it provides automated support for computing proposals for models that bring users closer to resolving their conflicts, allowing users to do "what if" exploration and choose the most suitable alternatives. We also keep a history of decisions that have been made, allowing users to study the results, and, if necessary, undo the decisions. Our methodology also guarantees that the negotiation process will eventually terminate, while inflicting only the minimal changes onto the original models.

The rest of this paper is organized as follows. Section 2 introduces our framework and present techniques for merging models and coping with inconsistency through exploration and selection of suitable resolutions. We discuss the implementation of our framework and preliminary evaluation in Section 3. Section 4 summarizes the paper and outlines venues for future work.

2. OVERVIEW OF THE APPROACH

In this section, we briefly describe our framework for model merging and negotiation. For further details, the reader can refer to the longer version [**?**] where the formal underpinnings of our approach have been elaborated.

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

2.2 The 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 2.2.1).

When models are inconsistent, their merge does not exist. In this case, we start the negotiation process that consist of three phases: **Computing Agreements**, **Exploration**, and **Resolution**. First, we build a model that reflects agreements between the initial models (**Computing Agreements**, Section 2.2.2). Effectively, we replace all points of potential contention with maybe. Afterwards, we project the result back onto the original models, allowing users to explore the result (**Exploration**, Section 2.2.3). Using global properties or their intuition, and supported by analysis tools such as model-checkers, users pick a list of items that they care about most. These become input to the **Resolution** phase (Section 2.2.4) which attempts to handle these disagreements by building consistent proposals, allowing users to pick their favourite. Once proposals have been chosen, the resulting models, which are now consistent but may still be somewhat incomplete, can be merged using techniques described in Section 2.2.1 (**Computing Merge**). The incompletenesses represent "don't cares" on the part of stakeholders, indicating that they will be satisfied with any consistent resolution of these points.

2.2.1 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 [28] 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* [25, 31] 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 are 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. A common refinement preserves common behaviours and temporal properties of the original models [25], and hence, is a reasonable candidate for the merge.

To construct a common refinement of two 3-valued state-machines M_1 and M_2, we first compute a consistency relation $\sim$ between states of M_1 and M_2. Intuitively, two states s and t are consistent, i.e. are related by $\sim$, iff they agree on the values of their propositions and their successors are consistent with each other as well. When models are consistent, such relation $\sim$ exists and can be computed automatically in polynomial time in the size of M_1 and M_2.

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.

2.2.2 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 ρ over the states of M_1 and M_2 that reflects the most similarities between these two models. The relation ρ is, in fact, analogous to the consistency relation $\sim$. As mentioned in Section 2.2.1, $\sim$ cannot be computed for inconsistent models. For this class of models, we map the states of the inconsistent models by a relation ρ in such a way that local inconsistencies between individual states become minimal.

Finding the best ρ 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 ρ 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. ρ, and further, $\hat{M}_1$ and M_2 as well as M_1 and $\hat{M}_2$ are pair-wise consistent with respect to ρ. 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.

2.2.3 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, as we discuss in Section 2.2.4, 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 (optional) **Exploration** phase is to choose the truly important items, which must be negotiated, from the overall list. We call this a *priority list (PL)*. Leaving an item off the PL indicates that the stakeholder is content with it becoming *either* t *or* f at some point in the future. Note also that each stakeholder builds her own PL independently, so if an item is really important to one stakeholder and not important to the other, the resolution is to simply choose the second user's value for this item.

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 this paper, we assumed that global properties are expressed in temporal logic, so any 3-valued model-checker, e.g., χChek [5], can be used for this analysis. Stakeholders may further prioritize those properties on which the analysis ended up being inconclusive, and restrict their PL just to those items that caused inconclusiveness of these most desirable properties. This information is also readily obtainable from a model-checking run. Specifically, χChek can return a counterexample explaining why the property evaluates to m. It also has a feature which returns *all* reasons why a property is m, in the form of an *abstract counterexample* [6]. For our framework, we modified χChek to extract the list of m variables and transitions from the returned counterexample and report its size to the user. This gives her an early indication about the feasibility of achieving agreements during the resolution step.

2.2.4 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 projection models $\hat{M}_1$ and $\hat{M}_2$, a consistency relation ρ between $\hat{M}_1$ and $\hat{M}_2$, and the priority lists of inconsistencies, obtained by merging the individual PLs. 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 ρ. Sometimes when ρ is too restrictive, the resolution algorithm may conflict with ρ and fails to resolve an m item. In this case, the tuples in ρ that cause the conflict are reported back to **Computing Agreements** phase (see Fig. 1), which attempts to find a better ρ, and then the resolution process is repeated again.

The complexity of resolving each individual item depends on the number of states or transitions affected by it, which is a (small) fraction of ρ. However, the number of generated proposals is exponential in the size of the joint list because the resolution of each item can potentially lead to two proposals. Fortunately, items in the priority list often depend on each other, effectively reducing the overall number of generated proposals. If the number of proposals turns out to be still large. We envision that in such situations, 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.

Proposals generated by the resolution algorithm are consistent with respect to ρ, just like the corresponding projections have been, but they refine these projections, deeming more properties conclusive. On the other hand, the relationship between proposals and the *original* models M_1 and M_2 is *orthogonal* to refinement: we first abstract from all inconsistencies, yielding maybes, and then refine the result consistently. For example, if a property φ was t in M_1 and f in M_2, it becomes m in $\hat{M}_1$ and $\hat{M}_2$, and, if present in the PL, is set either to t or to f in *both* $\hat{M}_1{}'$ and $\hat{M}_2{}'$. Thus, inconsistency resolution is *non-monotonic*!

As proposals are being generated, users can explore them and, if satisfied, accept one. The resulting models are guaranteed to give conclusive values to items on the priority list and be consistent with respect to ρ; thus, they can be easily merged. This is done in the **Computing Merge** phase (see Figure 1) and computed using the techniques described in Section 2.2.1.

3. TOOL SUPPORT AND PRELIMINARY EVALUATION

We have prototyped a proof of concept implementation of the merge and negotiation framework discussed in earlier sections. 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 χChek [5], allows stakeholders to explore its projections, building a priority list from χChek's abstract counterexamples. Once proposals are built (using the resolution algorithm), users can choose their favourite, apply suggested resolutions to their *original* models, and attempt to merge them again. This allows *incremental* negotiation. Our implementation supports it by additionally storing the history of made decisions, allowing users to go back and undo them (and thus facilitating "what if" exploration).

To try our framework on a more realistic example, we attempted to merge inconsistent descriptions of behaviour of an authentication system, adopted from [38]. The system is described from the administrator's and from the user's points of view, and the models disagree on a property "Entering a password can be followed by a successful authentication". The models were translated from MTSs [31] to 3-valued state machines and had 3 and 5 states, respectively, with the combined vocabulary of 3 variables. The size of ρ was 5, i.e., $max(\Sigma_1, \Sigma_2)$. We computed the maximal agreement model and automatically refined it using a heuristic discussed in [32]. Using the above-mentioned property, we identified only two priority items, which ended up being related. The resolution algorithm yielded two proposals, forcing the property into becoming `t` or `f` in both models, respectively. We note that one of the resulting proposals is exactly the modification of the user model done by hand in [38] in order to resolve inconsistencies, and that ρ is the same as the consistency relation obtained in [38]. Further details are available in [32].

4. CONCLUSION AND FUTURE WORK

In this paper, we have described a formal framework for merge and conflict resolution. This framework facilitates automatic merging of consistent models, enables users to visualize and explore potential disagreements and identify their priorities, and suggests ways to resolve the priority items.

Several research problems need to be solved to ensure that this framework is effective. The first and most important of these is the efficient computation of a relation ρ that reduces the size of the agreement models while capturing the maximal similarities between the inconsistent models. Computing an optimal ρ is similar to the *schema matching* problem – a subject that has been extensively studied in the database literature, e.g., [36]. We are currently looking for ways to tailor the existing schema matching techniques to our framework. We further need to evaluate the effectiveness of the framework on more realistic case-studies as well as develop additional heuristics to improve precision of the merge and a methodology for partitioning PLs in cases when the framework generates too many proposals.

Another direction is changing the resolution algorithm to produce more proposals. Our algorithm produces proposals obtained by keeping the relation ρ intact. More interesting proposals can be produced if we allow the algorithm to extend ρ. In this paper, we studied negotiation over flat state-machines. Adding hierarchy as well as more complex language features would enable merging and negotiation over more realistic models.

5. ACKNOWLEDGMENTS

We thank Mehrdad Sabetzadeh and Sebastian Uchitel for providing us with the examples and for their helpful comments. This research was partially funded by NSERC and MITACS.

6. REFERENCES

[1] N. Belnap. "A Useful Four-Valued Logic". In *Modern Uses of Multiple-Valued Logic*. Reidel, 1977.

[2] B. Boehm and H. In. "Identifying Quality-Requirement Conflicts". *IEEE Software*, 13(2), 1996.

[3] G. Bruns and P. Godefroid. "Generalized Model Checking: Reasoning about Partial State Spaces". In *CONCUR*, 2000.

[4] M. Chechik, B. Devereux, S. Easterbrook, and A. Gurfinkel. "Multi-Valued Symbolic Model-Checking". *ACM TOSEM*, 12(4), 2003.

[5] M. Chechik, B. Devereux, and A. Gurfinkel. "χChek: A Multi-Valued Model-Checker". In *CAV*, 2002.

[6] M. Chechik and A. Gurfinkel. "A Framework for Counterexample Generation and Exploration". In *FASE*, 2005.

[7] S. Clarke and R. J. Walker. "Composition Patterns: An Approach to Designing Reusable Aspects". In *ICSE*, 2001.

[8] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[9] J. Conklin and M. Begeman. "gIBIS: A Hypertext Tool for Exploratory Policy Discussion". *Trans. on Info. Sys.*, 4(6), 1988.

[10] D. Damian, A. Eberlein, M. Shaw, and B. Gaines. "An Exploratory Study of Facilitation in Distributed Requirements Engineering". *REJ*, 8(1), 2003.

[11] A. d'Avila Garcez, A. Russo, B. Nuseibeh, and J. Kramer. "An Analysis-Revision Cycle to Evolve Requirements Specifications". In *ASE*, 2001.

[12] S. Easterbrook. "Resolving Conflicts Between Domain Descriptions with Computer-Supported Negotiation". In *Workshop on Knowledge Acquisition for Knowledge Based Systems*, 1990.

[13] S. Easterbrook and M. Chechik. "A Framework for Multi-Valued Reasoning over Inconsistent Viewpoints". In *ICSE*, 2001.

[14] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. "Inconsistency Handling in Multi-Perspective Specifications". *IEEE TSE*, 20(8), 1994.

[15] 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.

[16] P. Godefroid and R. Jagadeesan. "On the Expressiveness of 3-Valued Models". In *VMCAI*, 2003.

[17] T. Gruber. "The Role of Common Ontology in Achieving Sharable, Reusable Knowledge Bases.". In *Principles of Knowledge Representation and Reasoning*, 1991.

[18] A. Gurfinkel and M. Chechik. "Multi-Valued Model-Checking via Classical Model-Checking". In *CONCUR*, 2003.

[19] A. Gurfinkel and M. Chechik. "Yasm: Model-Checking Software with Belnap Logic", 2005. Univ. of Toronto Tech. Report.

[20] D. Harel. "StateCharts: A Visual Formalism for Complex Systems". *Science of Computer Programming*, 8, 1987.

[21] M. Heimdahl and N. Leveson. "Completeness and Consistency in Hierarchical State-Based Requirements". *IEEE TSE*, 22(6), 1996.

[22] C. Heitmeyer, A. Bull, C. Gasarch, and B. Labaw. "SCR*: A Toolset for Specifying and Analyzing Requirements". In *COMPASS*, 1995.

[23] A. Hunter and B. Nuseibeh. "Managing Inconsistent Specifications: Reasoning, Analysis and Action". *ACM TOSEM*, 7(4), 1998.

[24] A. Hussain and M. Huth. "On Model Checking Multiple Hybrid Views". In *Inter. Symp. on Leveraging Apps. of FMs*, 2004.

[25] M. Huth, R. Jagadeesan, and D. A. Schmidt. "Modal Transition Systems: A Foundation for Three-Valued Program Analysis". In *ESOP*, 2001.

[26] M. Huth and S. Pradhan. "Model-Checking View-Based Partial Specifications". *Electr. Notes Theor. Comp. Sci.*, 45, 2001.

[27] D. Jackson. “Alloy: A Lightweight Object Modelling Notation”. *ACM TOSEM*, 11(2), 2002.

[28] S. C. Kleene. *Introduction to Metamathematics*. New York: Van Nostrand, 1952.

[29] D. Kozen. “Results on the Propositional μ-calculus”. *TCS*, 27, 1983.

[30] I. Krueger, R. Grosu, P. Scholz, and M. Broy. “From MSCs to Statecharts”. In *Conf. Distributed and Parallel Embedded Systems*, 1999.

[31] K. Larsen and B. Thomsen. “A Modal Process Logic”. In *LICS*, 1988.

[32] S. Nejati. “Negotiation for a B2B E-Commerce Site”. `http://www.cs.toronto.edu/~shiva/examples`.

[33] S. Nejati. “Abstraction and Software Model Checking”, 2005. Univ. of Toronto Depth Paper.

[34] C. Nentwich, W. Emmerich, A. Finkelstein, and E. Ellmer. “Flexible Consistency Checking”. *ACM TOSEM*, 12(1), 2003.

[35] B. Nuseibeh, J. Kramer, and A. Finkelstein. “Framework for Expressing the Relationship Between Multiple Views in Requirements Specifications”. *IEEE TSE*, 20(10), 1994.

[36] E. Rahm and P. A. Bernstein. “A Survey of Approaches to Automatic Schema Matching”. *VLDB Journal*, 10(4), 2001.

[37] M. Sabetzadeh and S. Easterbrook. “Analysis of Inconsistency in Graph-Based Viewpoints”. In *ASE*, 2003.

[38] S. Uchitel and M. Chechik. “Merging MTSs for a B2B E-Commerce Site”. `http://www.doc.ic.ac.uk/~su2/merge/examples`.

[39] S. Uchitel and M. Chechik. “Merging Partial Behavioural Models”. In *FSE*, 2004.

[40] S. Uchitel, J. Kramer, and J. Magee. “Synthesis of Behavioural Models from Scenarios”. *IEEE TSE*, 29(2), 2003.