Semantics-Aware Versioning Challenge: Merging Sequence Diagrams along with State Machine Diagrams

Petra Brosch
Business Informatics Group
Vienna University of Technology
Austria
brosch@big.tuwien.ac.at

Martina Seidl
Inst. for Formal Models and Verification
Johannes Kepler University Linz
Austria
martina.seidl@jku.at

Magdalena Widl
Knowledge-Based Systems Group
Vienna University of Technology
Austria
widl@kr.tuwien.ac.at

***Abstract*—In multi-view modeling languages like UML, models contain several diagrams, each of which focusing on a specific aspect of the system. However, when the diagrams are combined, they give a coherent description of all static and dynamic aspects of the system. Diagrams may then extend each other or add constraints to other diagrams. Considering this additional information improves model versioning, as conflicts are revealed also in case their changes are not overlapping, and merge algorithms may provide solutions which are correct by construction.**

This paper describes a *challenge benchmark* for semantics-aware merging of sequence diagrams with respect to their corresponding state machine diagrams.

I. Introduction

As software systems are getting larger, it is crucial to decompose its specification into components, different levels of abstraction, and several views, each focusing on specific static or dynamic aspects of the system [1]. Therefore, multi-view modeling languages like the *Unified Modeling Language* (UML) [2] provide several diagrams like *Class diagrams*, *State machine diagrams*, and *Sequence diagrams*, featuring a mutual relationship to each other. Altogether the diagrams render a coherent picture of the system. This paper targets the challenge and chance for model versioning systems to deal with the semantics of multi-view modeling languages. We focus on the evolution of sequence diagrams, which are associated with state machine diagrams. It is assumed that the state machine diagrams are stable and only the sequence diagrams change. In order to keep the scenario clear, we provide a dedicated metamodel for the *Tiny Multi-View Modeling Language* (*tMVML*). Although *tMVML* is inspired by UML, it hides unnecessary complexity and exhibits a compact representation of the concepts needed for our purpose. An excerpt of the *tMVML* metamodel is depicted in Figure 1. The root class of *tMVML* named Model contains the two classes StateMachineView and SequenceDiagramView representing views, and the class ActionSymbol, which realizes the relationship between those views. The StateMachineView contains a set of state machines, each containing a set of states. Transitions connect states and are associated with one action symbol as trigger and one or more action symbols as effect. The SequenceDiagramView contains a set of lifelines and a set of messages. Each lifeline is assigned to a state machine and contains a sequence of events, which are connected to a message. Messages carry an action symbol and are ordered relative to the lifelines they are attached to.

This work was supported by the Vienna Science and Technology Fund (WWTF) through project ICT10-018.

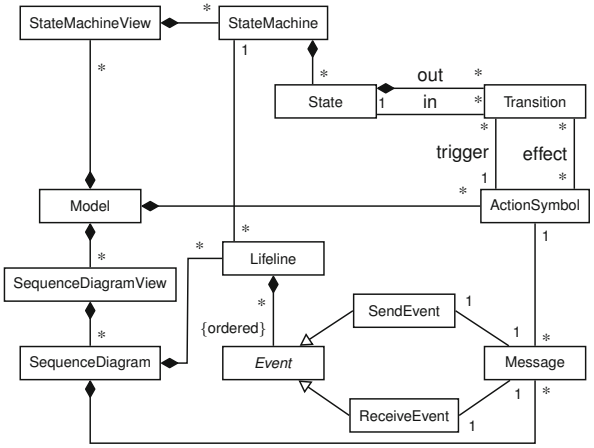

Fig. 1: Metamodel of the *tMVML*.

An Ecore based implementation of *tMVML* is available as Eclipse plug-in at our updatesite[1]. Further details on the formal semantics of tMVML are described in [3].

II. Example

The following example modeled with *tMVML* serves as motivation for semantics-aware model versioning. Figure 2 shows state machine diagrams describing the basic behavior of a PhD student and a coffee machine. *State machine diagrams* represent the system's behavior in terms of states the system may be in. States are connected

[1] http://www.modelevolution.org/updatesite/

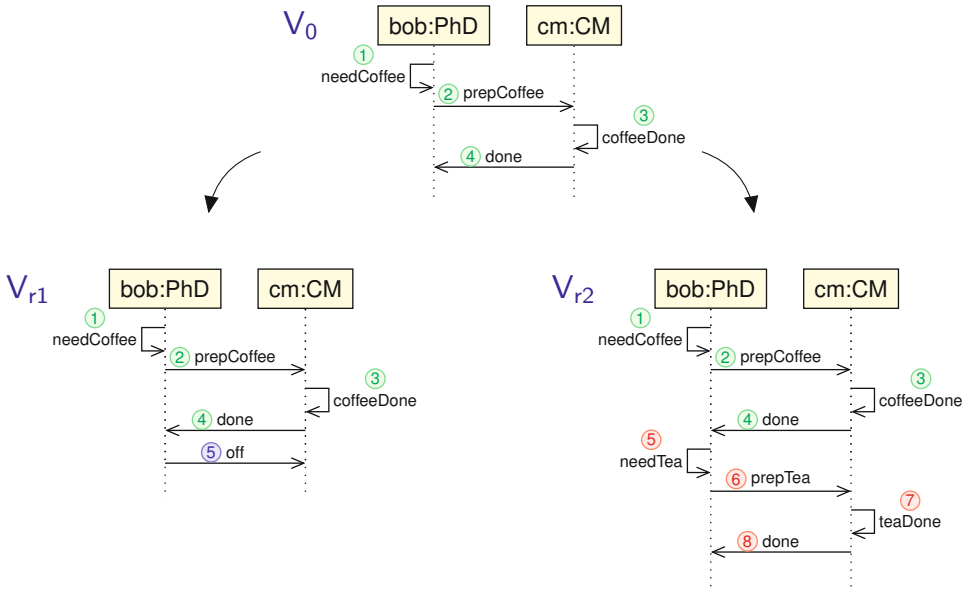

Fig. 3: Evolution of a sequence diagram.

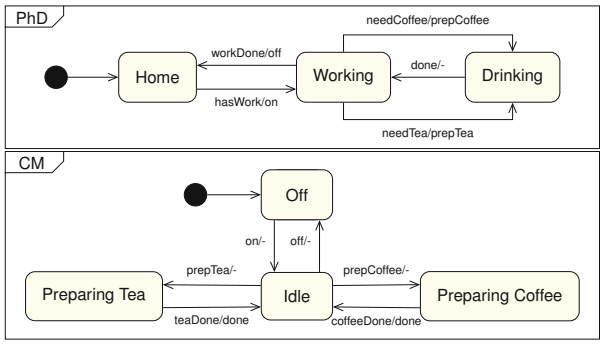

Fig. 2: State machine diagrams of a coffee machine and a PhD student.

to each other by transitions. The initial state of each state machine diagram is indicated by an incoming transition from a black circle. Each transition is labeled following the pattern trigger/effects. The *trigger* causes the state machine to change its state from the source state of the transition to the target state of the transition. The possibly empty set of *effects* consists of action symbols that are sent when the transition is executed and which may again trigger state transitions in the same or other state machines. For example, the state machine PhD of the Phd student starts in state Home and resides in this state until it receives hasWork, which causes a transition to state Working. When the transition is executed, it sends the effect on, which is received by the state machine CM of the coffee machine and leads to a transition from the initial state Off to state Idle.

Such communication scenarios are modeled by *sequence diagrams*. Communication partners are instances of state machines and are represented by *lifelines*, which exchange sequences of messages. In this work, we only consider synchronous message passing. Messages connect two lifelines and contain an action symbol. The action symbol glues the communications of the sequence diagram to paths of the state machine diagrams, as it may be found as (1) effect on some transition of the sender's state machine and (2) as trigger on some transition of the receiver's state machine. A sequence diagram is consistent with the state machine diagrams that are instantiated by its lifelines if for each lifeline the sequence of received messages is a path of triggers in the corresponding state machine. Figure 3 shows a communication scenario between bob, which is an instance of PhD and cm, an instance of CM. The communication depicted in the uppermost diagram of Figure 3 named V_0 is as follows. The sequence of received messages in lifeline bob:PhD causes the triggers needCoffee → done, which may be found as path in the state machine PhD, namely Working → Drinking → Working.

Consider the versioning scenario depicted in Figure 3. The uppermost sequence diagram V_0 is the head revision in a versioning repository. Two modelers, let us call them Harry and Sally, check out this version and start revising the sequence diagram. Harry adds the message off to the sequence diagram and commits his new revision depicted in V_{r1} of Figure 3 to the repository. In the meantime Sally is also working on the sequence diagram. She adds a sequence of messages for preparing tea after the existing messages, as shown in V_{r2} of Figure 3. When she commits her work to the repository, a generic versioning system does not report any conflict, as changes to multivalued references like occurred when adding new

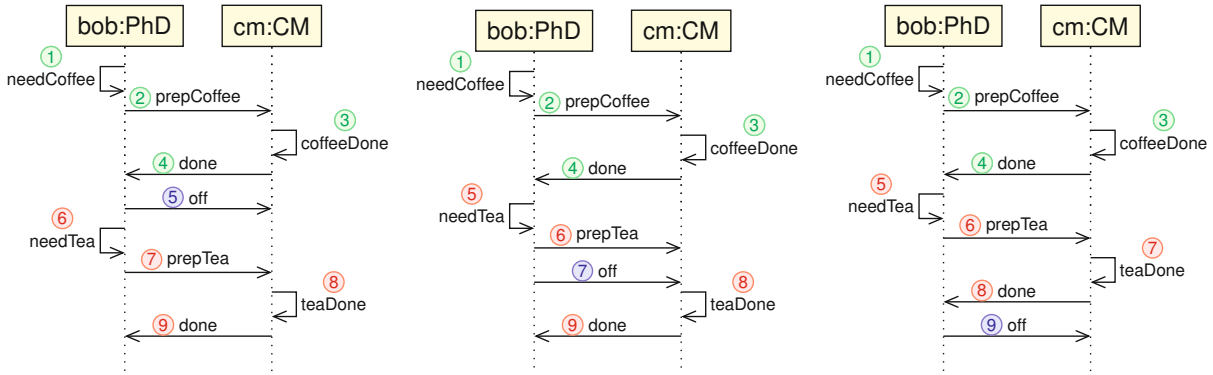

Fig. 4: Three time consistent, but not necessarily lifeline conformant, merges.

events to the lifelines are ignored and the modeling language's semantics is neglected.

III. Challenge

The challenge to overcome for solving the example described above is to find a consolidated version, which contains all original and changed messages and lifelines, is valid with respect to the tMVML metamodel, and is *correct*, i.e., it fulfills the following properties. (1) A sequence diagram is *well-formed*, if for each lifeline the order of events is total. This total order of events per lifeline imposes an ordering relation for messages. (2) A sequence diagram is *time consistent*, if any message m is received before message n if m has been sent before n on the same lifeline. (3) A lifeline is *conformant* with its state machine, if the sequence of action symbols carried by the messages received by the lifeline occurs as path of triggers in the state machine.

A consolidated version should further maintain the relative ordering of messages in the origin sequence diagram and in its two revisions. In case messages are added to the same relative position in both revisions, any interleaving of those messages is allowed, which is time consistent and maintains the original ordering. Figure 4 shows three possible orderings for the evolution scenario depicted in Figure 3. Message off of V_{r1} may interleave the added message sequence of V_r2 at any position, while the relative position with respect to the original messages is maintained. However, only the rightmost diagram is consistent with the state machine. The computational complexity of finding a consolidated version is thus given by the exponential number of message orderings and the lifeline conformance property.

Multi-view model versioning expose new challenges to existing differencing and merge algorithms, rendering generic solutions almost impossible. Additional effort is necessary, as the semantics of the targeted modeling language must be formalized. However, the strength of semantics-aware versioning is evident. (1) Conflicts arising even due to non-overlapping changes are detected if the lifeline conformance property is not satisfiable and (2) merge algorithms may provide consolidated versions, which are correct by construction.

All models and metamodels relevant to the described example are available for benchmarking purposes in the conflict lexicon Colex [4].

References

[1] M. Broy, "Multi-view Modeling of Software Systems," in *Formal Methods at the Crossroads. From Panacea to Foundational Support*, ser. LNCS, B. Aichernig and T. Maibaum, Eds. Springer, 2003, vol. 2757, pp. 207–225.

[2] O. M. Group, "OMG Unified Modeling Language (OMG UML), Superstructure V2.4.1," http://www.omg.org/spec/UML/2.4.1/, August 2011.

[3] M. Widl, A. Biere, P. Brosch, U. Egly, M. Heule, G. Kappel, M. Seidl, and H. Tompits, "Guided Merging of Sequence Diagrams," in *Software Language Engineering*, ser. LNCS, K. Czarnecki and G. Hedin, Eds. Springer, 2013, vol. 7745, pp. 164–183.

[4] C. Community, "Add Message to Sequence Diagram—Colex, The Conflict Lexicon," http://modelversioning.org/colex/conflicts/show/56, 2013, [Online; accessed 2013-02-14].