On the Challenges of Composing Multi-View Models

Matthias Schöttle and Jörg Kienzle
School of Computer Science, McGill University, Montreal, Canada
mschoettle@cs.mcgill.ca, joerg.kienzle@mcgill.ca

Abstract. The integration of compositional and multi-view modelling techniques is a promising research direction aimed at extending the applicability of model-driven engineering to the development of complex software-intensive systems. This paper outlines a general strategy for extending or integrating existing compositional modelling techniques into a multi-view approach. We demonstrate the practicality of our idea by explaining how we extended the Reusable Aspect Models (RAM) approach, which originally only supported structural modelling using class diagrams, with additional behavioural views based on sequence diagrams. This involved the integration of the metamodels as well as the model weavers.

1 Introduction

Model-Driven Engineering (MDE) [6] is a unified conceptual framework in which software development is seen as a process of *model production*, *refinement* and *integration*. Modelling is most effective when the properties of the system under development are modelled using the most appropriate modelling notations to express the properties of interest at the right level of abstraction. Models of the *same* system expressed in different modelling notations are called *views*. The ultimate goal of MDE is to obtain an executable model, e.g., in the form of source code. Therefore, it is necessary to integrate the different modelling notations that are used to describe the system, and to ensure that the different views are consistent with each other.

Models of complex systems tend to grow in size, to a point where even individual views are difficult to understand or analyse. To reduce model complexity, model composition mechanisms have been proposed that allow modellers to combine several models of the same modelling notation into one [1]. While these mechanisms can be readily applied to compose models expressed in the same notation, they cannot be applied as such within multi-view modelling approaches.

In this paper we outline a general approach for adapting existing composition mechanisms into a multi-view modelling context. The paper is structured as follows: Section 2 describes in general how to integrate an additional compositional modelling notation with an existing one to form a compositional multi-view modelling approach. Section 3 shows how we applied this idea in the context of the Reusable Aspect Models (RAM) [4] approach, which originally only supported structural modelling with class diagrams. We explain how we added an additional behavioural view that is based on sequence diagrams to the metamodel, and how we integrated the class diagram and sequence diagram weavers to correctly handle multi-view models. Finally, the last section draws some conclusions.

2 Integration Strategy

This section describes a general strategy for integrating existing compositional modelling techniques into a multi-view approach. Our idea can also be applied when an existing compositional modelling approach is to be extended with an additional view. For the sake of clarity of the discussion, we describe in this section how two modelling notations I and D are integrated into a multi-view modelling notation I/D. We further assume that at least the metamodel and composer for I of the modelling notations are defined.

2.1 Integrating the Metamodels

The first step is to create a new metamodel that integrates I and D. To ensure that the two views are consistent with each other, it is important to unify the elements that represent concepts that are shared between I and D. The main idea of our strategy is to leave one of the metamodels untouched. We call this metamodel the *independent metamodel* MM_I. The second metamodel, named the *dependent metamodel* MM_D, is modified (or created) in such a way that it builds on the first one, i.e., it references or reuses metamodel elements from MM_I for all conceptually shared concepts.

Modifying the second metamodel can be as simple as changing a reference to a metaclass DC in MM_D to refer to the metaclass IC in MM_I that represents the same shared concept. However, it can also involve more intricate changes. For example, a behavioural modelling notation could specify an action that is executed at a certain point in time using text: "transferMoney(a,b,100)" could signify that at a certain point an operation is invoked that transfers 100 dollars from account a to b. If this behavioural modelling notation is being integrated with a structural modelling notation that defines operations, then this text attribute should be replaced by a reference to the metaclass in the structural metamodel that represents an operation.

2.2 Updating the Model Composers

The metamodel MM_I was left untouched, and therefore the existing composer MC_I still works as is, i.e., it can compose two models I_1 and I_2 that are instances of MM_I to produce a composed model I_C. However, we additionally require that MC_I provides tracing information that details for each model element in I_1 and I_2 to which element(s) in I_C they were mapped to. Note that composers that support tracing typically already provide such information.

Since MM_D was modified, the composer MC_D must be adapted to work with the new metamodel. This adaptation is rather straightforward, since the composition algorithm does not change. Only the parts that deal with the composition of the metamodel elements that now refer to MM_I need to be updated.

2.3 Composition Algorithm

This subsection describes an algorithm that uses the original composer MC_I and the modified composer MC_D to compose two multi-view models M_1 and M_2. As depicted in the upper left corner of Fig. 1, each source model consists of a part expressed using instances of MM_I (shown as I_x) and a part using instances of

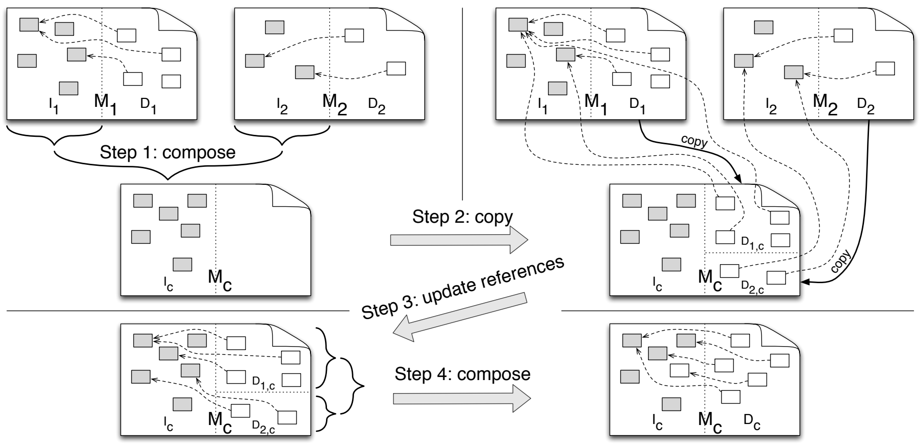

Fig. 1. Composition of Two Multi-View Models

MM_D (shown as D_x). Since MM_D refers to MM_I, some instances in D_x refer to instances in I_x. This is depicted using directed dashed lines.

The multi-view composition algorithm proceeds in 4 steps:

1. First, I_1 and I_2, i.e., the parts of the models that are expressed with the independent metamodel MM_I, are composed using the unmodified composer MC_I. This outputs a composed model M_C, which so far contains the elements in I_C (see result of step 1 in upper left part), as well as tracing information on how elements in I_1 and I_2 were mapped to elements in I_C.
2. Next, the elements from D_1 and D_2 are copied into M_C (see step 2 in Fig. 1). The result is an inconsistent model M_C, because it contains external references (model elements in $D_{x,c}$ still refer to elements in I_x of M_x).
3. The tracing information of step 1 is then used to remove the external references to I_1 and I_2, and replace them with references to the corresponding elements in I_C (see step 3 in Fig. 1). As a result, M_C is now internally consistent, but still contains uncomposed elements ($D_{1,C}$ and $D_{2,C}$).
4. Finally, the modified composer MC_D is invoked on $D_{1,C}$ and $D_{2,C}$ to yield the final composed model M_C (see step 4 in Fig. 1).

3 Integrating Message Views Into RAM

Reusable Aspect Models (RAM) [4] is a compositional multi-view modelling approach for concern-oriented software design. On paper, a RAM model supports structural modelling using class diagrams *(structural view)*, behavioural modelling with sequence diagrams (*message views*), and protocol modelling with state diagrams (*state views*). However, the TouchRAM tool [2] until recently only supported structural modelling. This section describes how the general integration strategy described above was used in the context of TouchRAM to integrate a compositional behavioural modelling notation with an already existing compositional structural modelling notation.

Integrating the Metamodel and Updating the Composer: The metamodel of message views (MM_{MV}), shown in Fig. 2, is loosely based on UML

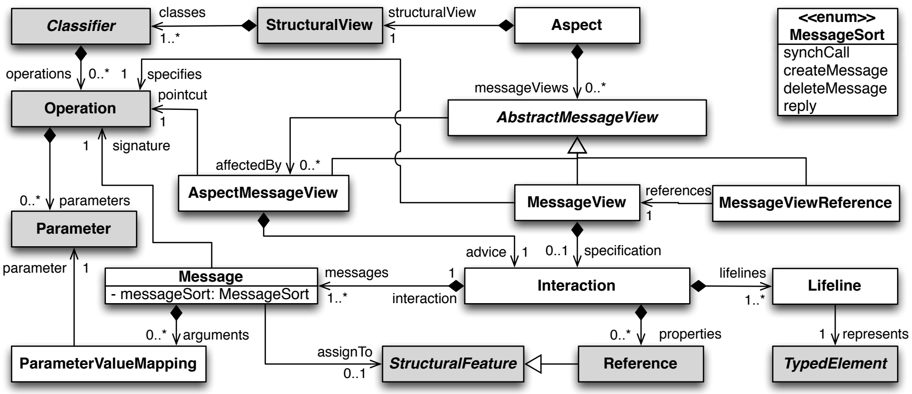

Fig. 2. MessageView (white) integrated with the StructuralView Metamodel (grey)

Sequence Diagrams [5]. It is heavily simplified, though, as UML supports different types of interaction diagrams. Furthermore, following the strategy outlined in section 2, we left the RAM structural view metamodel (MM_{SV}) untouched, and made sure that the concepts in MM_{MV}, that also appear in MM_{SV}, were implemented as references to MM_{SV}. For example, a RAM *Message* refers to an *Operation* in the structural view describing which operation is invoked, which in turn has a name and parameters. This is different from UML, where it is possible to supply a textual description as a message signature. The fact that MM_{SV} is the independent metamodel (MM_I) and MM_{MV} the dependent one (MM_D) is nicely illustrated in Fig. 2: The only associations between message view elements (white) and structural view elements (grey) are directed from white to grey.

We also changed the structural weaver to generate a "weaving information" data structure that remembers for each element in the two input models to which output model element it was mapped to.

RAM Composition Algorithm: In RAM, a model M_1 can depend on a model M_2 to implement low-level design details. For that, M_1 contains instantiations that map generic model elements of M_2 to model elements in M_1.

Whenever two RAM models M_1 and M_2 are composed, structural weaving is performed first (step 1), combining the structural views to produce a woven model M_C with the composed structure. This produces weaving information with the detailed composition mapping. Then, all message views from the lower-level model M_2 are copied into M_C (step 2). At this point, the message views still refer to structural elements in M_2. Therefore, using the weaving information, all references are updated to refer to the right elements in M_C (step 3).

While M_C already is a consistent model, the behaviour is still uncomposed. If desired, the RAM modeller can instruct TouchRAM to collect all message exchanges of a given behaviour within one message view (step 4). This is called *message view inlining*, and, when performed, results in replacing all operation invocations within a message view with the scenarios specified in the corresponding message views. This shows the modeller all behaviour that is executed when a given operation is invoked in one place.

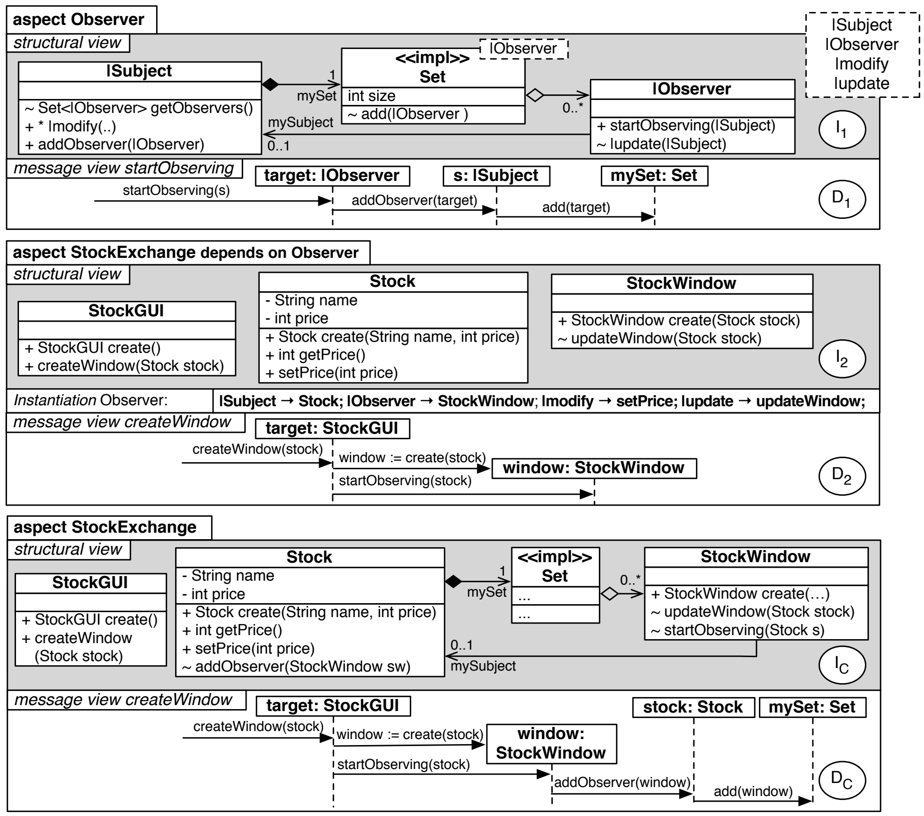

Fig. 3. The *Observer StockExchange* and woven *StockExchange* Models

3.1 Composition Example

The *Observer* RAM model shown in the top part of Fig. 3 describes the observer design pattern. Whenever a *|Subject* is modified (i.e., the operation *|modify* is called), it notifies all its *|Observers* by calling their *|update* operation.

The *Observer* model is reused by *StockExchange*, which is depicted in the middle of Fig. 3. *StockExchange* defines a *Stock* class with a name and price. A *StockWindow* displays the information of a specific stock. The *StockGUI* is responsible for creating the overall user interface, i.e., windows displaying each stock. In order to reuse the *Observer* aspect, *StockExchange* has to map all partial classes of Observer (i.e., classes with a "|" prefix)to elements in *StockExchange*: *|Subject* is mapped to *Stock* and *|Observer* to *StockWindow*. Operations that modify the stock include the setter for the attribute *price* (setPrice); *updateWindow* is the operation that refreshes the window with the updated information of a stock.

The result of applying our composition algorithm is shown at the bottom of Fig. 3. The composition of the *structural view* merges all classes that are mapped in the instantiation directive, and copies the unmapped classes from the lower- to the higher-level aspect. For example, *Stock* and *|Subject* are merged, but *Set* is copied from *Observer* to *StockExchange*. After the structural view is composed, all *message views* are copied from *Observer* to *StockExchange* and the references updated to point to the classes of the *StockExchange* aspect. This results in a functional woven aspect with no dependencies. If the modeller subsequently wishes to see the complete behaviour of *createWindow*, she can perform *message view inlining* on this operation, which results in the message view *createWindow* shown at the very bottom of Fig. 3.

4 Conclusion

This paper presented a strategy for integrating existing compositional modelling techniques into a multi-view approach(or, alternatively, for extending an existing technique with an additional view). We demonstrated the practicality of our strategy by extending TouchRAM, which supported compositional structural modelling with structural views (class diagrams), with behavioural views expressed using message views (sequence diagrams). The interested reader is referred to [7] for a detailed description of the integration of message views.

How well the metamodels can be integrated depends heavily on how well the concepts from the independent metamodel MM_I are aligned with the concepts of MM_D. If the level of detail of MM_I is higher (i.e. the mapping from MM_D to MM_I is one to many), it might be possible to add a new superclass into MM_I. The composer must then be heavily updated. For example, we had to introduce a superclass *TypedElement* for all elements with a type into the structural view metamodel in order to support message views. If the concepts in MM_I are more general, then it might be possible to add new subclasses into MM_I without affecting already existing models or the composer MC_I.

Based on our experience, we believe that our strategy is general, i.e., it can be applied to any compositional multi-view modelling approach, but further research has to be conducted to prove this. We are, however, confident, because we also used our strategy to successfully integrate state diagrams into TouchRAM, which is described in detail in [3].

References

1. Aspect-Oriented Modeling Workshop Series, `http://www.aspect-modeling.org/`
2. Al Abed, W., Bonnet, V., Schöttle, M., Alam, O., Kienzle, J.: TouchRAM: A Multitouch-Enabled Tool for Aspect-Oriented Software Design. In: SLE 2012. pp. 275 – 285. No. 7745 in LNCS, Springer (2012)
3. Ayed, A., Kienzle, J.: Integrating Protocol Modelling into Reusable Aspect Models. In: Proceeding of BM-FA 2013. pp. 1–12. ACM (July 2013)
4. Kienzle, J., Al Abed, W., Klein, J.: Aspect-Oriented Multi-View Modeling. In: AOSD '09. pp. 87–98. ACM, New York, NY, USA (2009)
5. Object Management Group: OMG Unified Modeling Language (OMG UML), Superstructure (August 2011), `http://www.omg.org/spec/UML/2.4.1/`, Version 2.4.1
6. Schmidt, D.C.: Guest Editor's Introduction: Model-Driven Engineering. IEEE Computer 39(2), 25–31 (2006)
7. Schöttle, M.: Aspect-Oriented Behavior Modeling In Practice. M.Sc. Thesis, Department of Computer Science, Karlsruhe University of Applied Sciences (September 2012), `http://mattsch.com/papers/masterthesis.pdf`