Modeling Aspect Weaving

Wisam Al Abed[1]

School of Computer Science, Mcgill University,
Montreal, QC, Canada,
wisam.alabed@mail.mcgill.ca

Abstract. This paper discusses an approach by Kienzle et al.[1] to modeling aspects within the context of transactional frameworks. It touches upon the aspect-oriented modeling (AOM) and feature modeling techniques used. The AOM approach allows the definitions of stand-alone reusable aspect models, and supports the modeling of structure(using UML class diagrams) and behavior (using UML sequence diagrams). The feature modeling approach allows the handling of variabilities in the product line built with several aspect models. We discuss the notion of inter-aspect weaving, which allows an aspect to reuse functionality modeled by other aspects. The resulting aspect model can be woven with an application model in order to create a structural and behavioral model of the application augmented with the functionality modeled by the aspect.

1 Introduction

In order to use aspect-oriented modeling in a viable manner, there is a need to introduce a notion of weaving at the modeling level. The motivation for this, as the paper [1] points out, is to allow for simpler and potentially more intuitive ways of validation, simulation and code generation of the system that is being modeled. The paper[1] presents an approach where any concern or functionality that is reusable is modeled as an aspect: the functionality defined within the aspect cuts across all the applications in which the functionality is reused. Hence, the approach presented allows the modeler to specify the structure and behavior of a concern in a reusable aspect model. Furthermore, the paper[1] applies this to the transactions domain for the sake of having a non-trivial and interesting example to work with. In this paper, We shall touch a little bit upon this approach by examining some of the aspects from the Aspect Optima transactions Framework. The paper is structured as follows: Section 2 introduces the example we shall work with as well as the transaction product line of the transactions framework presented in the original paper[1]. We shall also introduce the naive approach at modeling this example within a particular transactions configuration. Section 3 will introduce the aspect-oriented approach of modeling the example as presented in the paper[1]. This will be restricted to looking at two simple concerns in the transactions framework. Section 4 will summarize the results and conclude.

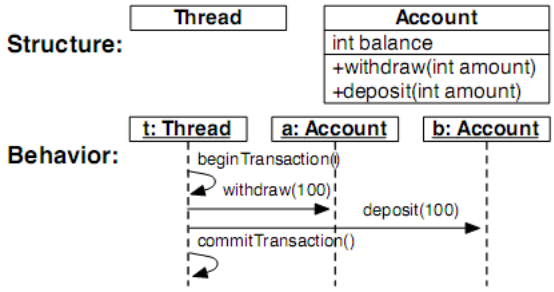

Fig. 1. A simple Application Model[1]

2 The Naive Approach

Fig.1 defines a simple application model of a bank, where a Thread instance t transfers 100 from account a to account b by calling a's withdraw method followed by b's deposit method. The transfer is enclosed within beginTransaction and commitTransaction invocations. We wish to apply some transactional properties to this application. Fig.2 defines a transaction product line proposed by Kienzle et al.[1]. Each feature can be thought of as a separate concern for a particular transaction that have dependencies amongst one another as portrayed in the feature diagram. To make this more concrete we can, for example, choose particular features in order to create a certain configuration. Fig.3 shows this for a flat transaction configuration. Suppose we wish to have the concerns shown in Fig.3 be applied to our simple bank application. One possibility is that we could, by hand, construct the structure and behavior of this application to conform to a flat transaction; this would involve large amounts of effort and brings forth an increased risk in introducing errors. This approach is also not very modular, hence limiting reusability. Fig.4 and Fig.5 is an example of the structure and behavior after applying the flat transaction to our bank application. It is clear that this is quite complex. As the application increases in complexity one can imagine that this approach will explode exponentially. There must be a better way for us to modularize this by potentially breaking the construction into smaller pieces and then defining rules for how to combine things together in order to come up with the final weaved model. This is what shall be discussed in the next section regarding the aspect-oriented approach presented in the paper[1] to tackle this issue.

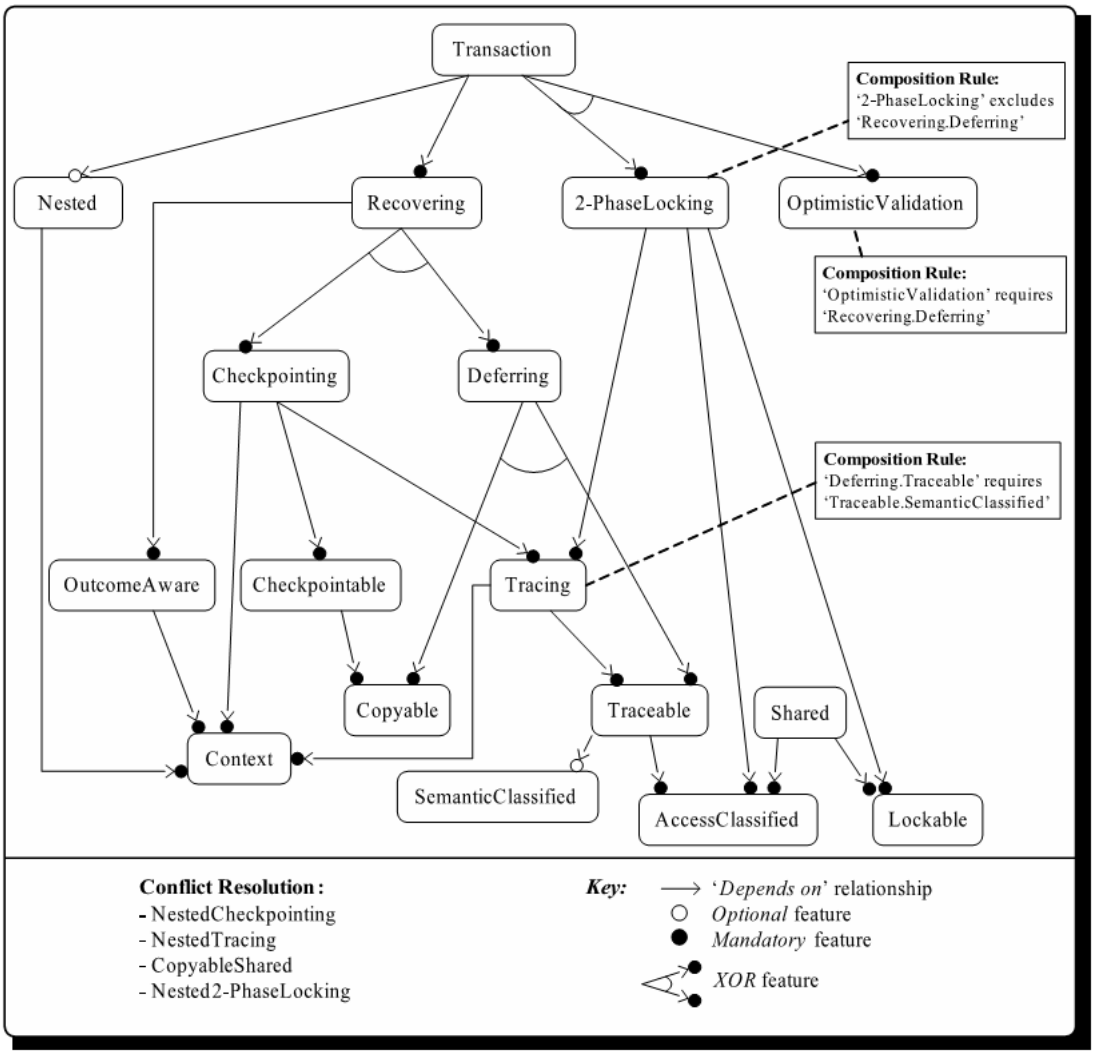

Fig. 2. The product line of transactions[1]

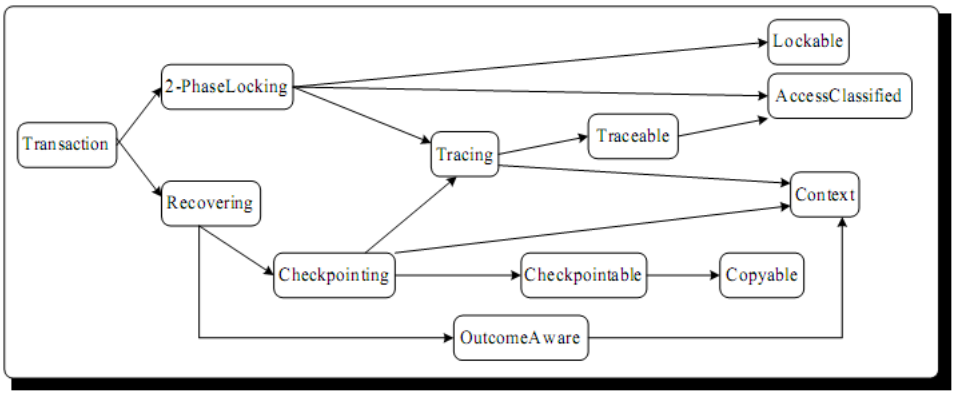

Fig. 3. Aspect Dependency chain for a Flat Transaction[1]

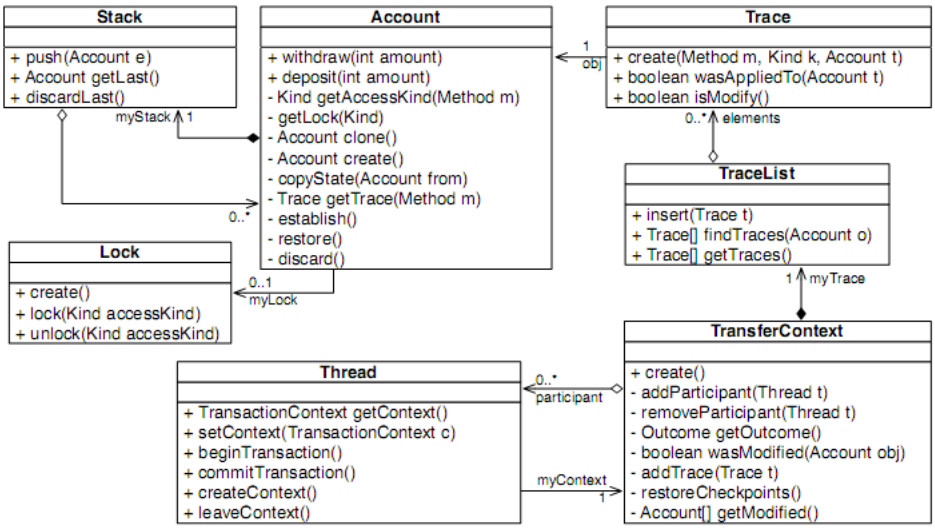

Fig. 4. Structure of Woven Application Model[1]

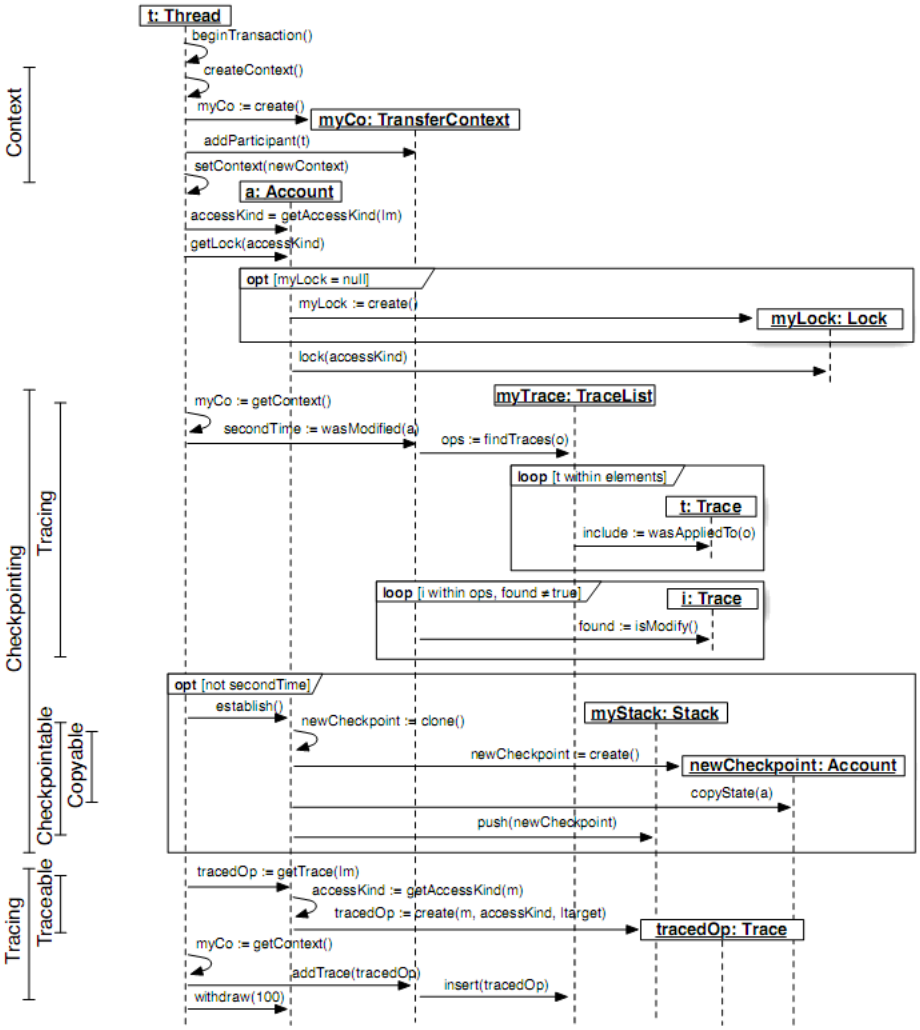

Fig. 5. Behavior of Woven Application Model[1]

3 The Aspect Oriented Approach

The paper[1] presents an aspect-oriented approach that allows one to define the structure and behavior of a concern in a reusable aspect model. Then using feature modeling or product line concepts it combines these two ideas to allow a modeler to choose the concerns he desires from the product line resulting in a certain configuration. This is doable since the aspects are reusable and composable as presented in the paper and then these inter-dependent aspects will be weaved together resulting in a final product. So we have a way of creating aspects that are reusable. These aspects can depend on one another creating potential dependency trees. We shall now show two examples of aspects, taken from the Aspect Optima Framework, that outline the modeling notation presented by Kienzle et al.[1].

An Example: Traceable and AccessClassified

An aspect is defined within a UML package identified with the keyword aspect. The aspect structure is defined in a compartment with the keyword structure, followed by behavior compartments for each functionality that the aspect provides that requires message exchanges among objects. If no message exchanges between objects are necessary, then a simple method declaration in the structure compartment suffices. Each behavior is split into two parts with the keywords Pointcut and Advice.

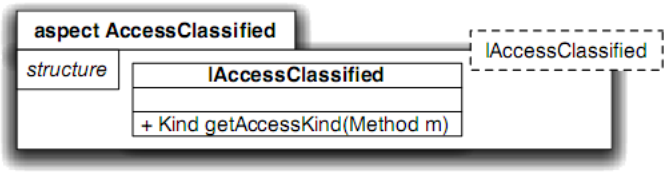

Fig. 6. The *AccessClassified* Aspect[1]

Fig.6 shows the AccessClassified aspect. It has no behavior only structure and simply specifies a class with one method. The paper proposes to use UML template parameters in order to identify within an aspect model which modeling elements are generic. Reusing an aspect model involves binding the aspect models template parameters to target model-specific elements. In the example in Fig. 6, the class |AccessClassified is a template parameter; when wanting to bind this we can declare instantiation directives. For example the following: $AccessClassified.|AccessClassified \rightarrow Transactional*$ specifies that all classes that have names starting with the string Transactional... are extended with the functionality defined in AccessClassified. This is important since once

this is declared successfully we get a context-specific aspect model that can be composed with a target model.

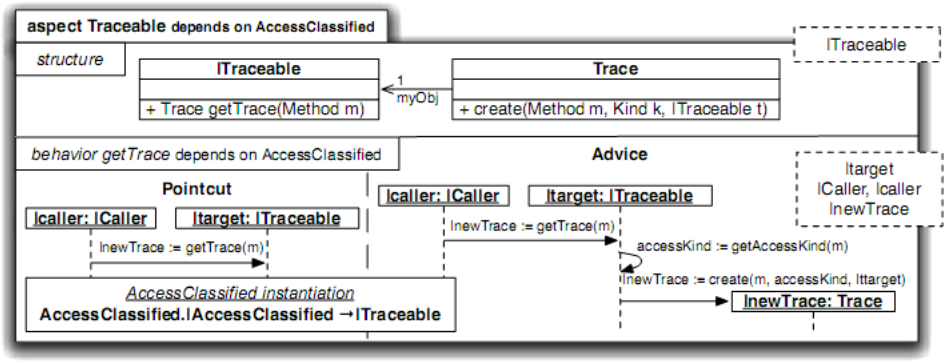

Fig. 7. The *Traceable* Aspect[1]

Fig.7 shows the Traceable aspect which depends on AccessClassified. It contains structure and behavior as well as an instantiation directive that allows us to reuse the functionality in AccessClassified. The instantiation directive within getTrace, $AccessClassified.|AccessClassified \rightarrow |Traceable$, declares that all Traceable objects have to be AccessClassified as well. This is an example of how dependency amongst aspects is handled in the approach presented by Kienzle et al.[1]. This is also important in demonstrating the reusability of the aspects using this approach. Now that we have demonstrated how these aspects get modeled we shall examine how they are weaved. Traceable depends on AccessClassified and we have seen how the instantiation directives define this. So before Traceable can be woven into an application model, AccessClassified must first be woven into Traceable creating an *independent model*. This weaving is like any other weaving. After binding the template parameter of AccessClassified according to the instantiation directive shown above, we attempt to match the pointcut of AccessClassified against the advice of Traceable. Then, any occurrences of the pointcut within the advice of Traceable are composed with the advice of AccessClassified. Of course in this case this is trivial since AccessClassified does not define any behavior, but this technique is a general technique that can be applied to any two aspects that are being woven into one another.

4 Conclusion

In this paper, we have presented a brief and simplified overview on how Kienzle et al.[1] built an aspect-oriented modeling framework. We also touched

upon how they combined feature modeling techniques with their AOM approach, as a way of managing variability in their framework. We discussed how the derivation of a transaction is performed by weaving the inter-dependent aspect models involved in the aspect dependency chain obtained from the product line. We have also shown how their weaver of reusable aspect models supports the modeling of structure (using UML class diagrams) and behavior (using UML sequence diagrams). Of course their paper touches upon many other issues but our concern for the future is o take this small subset and attempt to implement it, in order to act as a proof of concept, using a meta-modeling and graph rewriting tool.

References

1. Kienzle, J., Klein, J., Guel, N.: Modeling Aspect-Oriented Framework: Handling Aspect Reusability, Dependencies, Variabilities and Conflicts. Transactions on Aspect Oriented Software Development (TAOSD) (2008)